JAPANESE PATENT OFFICE

PATENT ABSTRACTS OF JAPAN

(11)Publication number: 10091443
(43)Date of publication of application: 10.04.1998

(51)Int.Cl.
G06F 9/42
G06F 9/30
G06F 9/46

(21)Application number: 09135923
(22)Date of filing: 08.05.1997

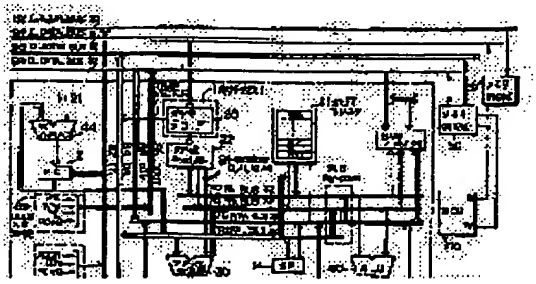
(71)Applicant: SEIKO EPSON CORP
(72)Inventor: KUBOTA SATORU
KUDO MAKOTO
MIYAYAMA YOSHIYUKI
SATO HISAO

(30)Priority
Priority number: 08127541 Priority date: 22.05.1996 Priority country: JP

(54) INFORMATION PROCESSING CIRCUIT, MICROCOMPUTER AND ELECTRONIC EQUIPMENT

(57)Abstract:

PROBLEM TO BE SOLVED: To efficiently describe a processing for dealing with a stack pointer with short instruction length, to efficiently describe the processings of execution, register saving and register restoration and to improve the processing speed of an interruption processing and sub-routine call return. SOLUTION: CPU 10 decodes a stack pointer-only instruction group which contains a register SP14 only for the stack pointer and sets SP14 to be a mute operand by an instruction decoder 20. The stack pointer-only instruction group is executed in terms of



hardware by using a generalpurpose register 11, PC12, SP14, an address adder 30, ALU 40 a PC incrementor 44, internal buses 72, 74, 76 and 78, internal signal lines 82, 84, 86 and 88 and external buses 92, 94, 96 and 98. The stack pointer-only instruction group contains an SP relative load instruction, a stack pointer shift instruction, a call instruction, a return instruction, a continuous push instruction and a continuous pop instruction.

LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's decision of rejection]

[Date of extinction of right]

(19)日本国特許庁（ＪＰ）　(12)公開特許公報（Ａ）　(11)特許出願公開番号

特開平10－91443

(43)公開日　平成10年(1998)４月10日

(51)Int.Cl.[6]		識別記号	FI		
Ｇ０６Ｆ	9/42	３３０	Ｇ０６Ｆ	9/42	３３０Ａ
	9/30	３５０		9/30	３５０Ｂ
	9/46	３１３		9/46	３１３Ｚ

審査請求　未請求　請求項の数17　ＦＤ　（全 29 頁）

(21)出願番号　特願平9－135923

(22)出願日　平成９年(1997)５月８日

(31)優先権主張番号　特願平8－127541

(32)優先日　平８(1996)５月22日

(33)優先権主張国　日本（ＪＰ）

(71)出願人　000002369
セイコーエプソン株式会社
東京都新宿区西新宿２丁目４番１号

(72)発明者　久保田　哲
長野県諏訪市大和３丁目３番５号　セイコーエプソン株式会社内

(72)発明者　工藤　真
長野県諏訪市大和３丁目３番５号　セイコーエプソン株式会社内

(72)発明者　宮山　芳幸
長野県諏訪市大和３丁目３番５号　セイコーエプソン株式会社内

(74)代理人　弁理士　井上　一　（外２名）

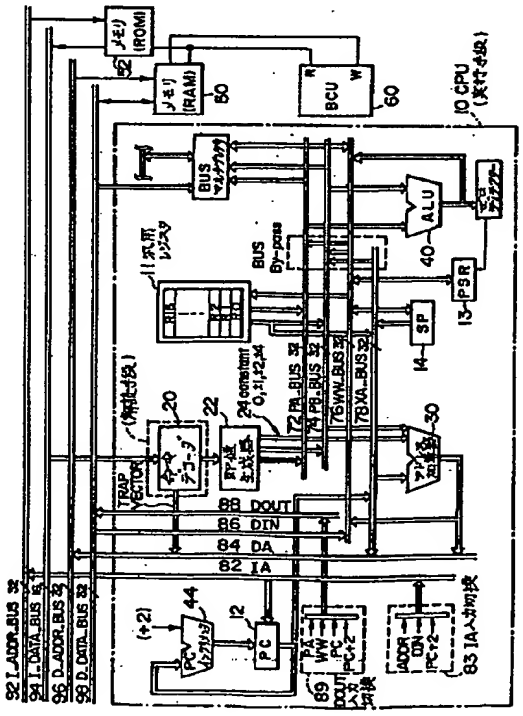
最終頁に続く

(54)【発明の名称】　情報処理回路、マイクロコンピュータ及び電子機器

(57)【要約】

【課題】　スタックポインタを取り扱う処理を、短い命令長で効率よく記述し、実行すること及びレジスタ退避やレジスタ復旧の処理を、効率よく記述し、割り込み処理及びサブルーチンコール・リターンの処理速度を向上させること。

【解決手段】　ＣＰＵ１０は、スタックポインタ専用のレジスタＳＰ１４を含みＳＰ１４を暗黙のオペランドとするスタックポインタ専用命令群を命令デコーダ２０で解読する。そして、汎用レジスタ１１、ＰＣ１２、ＳＰ１４、アドレス加算器３０、ＡＬＵ４０、ＰＣインクリメンタ４４、内部バス７２、７４、７６、７８、内部信号線８２、８４、８６、８８、外部バス９２、９４、９６、９８等を用いて前記スタックポインタ専用命令群をハードウエア的に実行する。前記スタックポインタ専用命令群は、ＳＰ相対ロード命令やスタックポインタ移動命令やｃａｌｌ命令やｒｅｔ命令や連続プッシュ命令や連続ポップ命令を含む。



1

【特許請求の範囲】

【請求項1】　スタックポインタ専用に用いるスタックポインタ専用レジスタと、

該スタックポインタ専用レジスタを暗黙のオペランドとするオブジェクトコードを有し、該スタックポインタ専用レジスタに基づく処理が記述されたスタックポインタ専用命令群のオブジェクトコードを解読して、該オブジェクトコードに基づき制御信号を出力する解読手段と、

前記スタックポインタ専用命令群を、前記制御信号及び前記スタックポインタ専用レジスタの内容に基づき実行する実行手段とを含むことを特徴とする情報処理回路。

【請求項2】　請求項1において、

前記スタックポインタ専用命令群が、転送レジスタ特定情報をオブジェクトコードに有するロード命令を含み、

前記解読手段が、

前記ロード命令を解読し、

前記実行手段が、

前記ロード命令を実行する際、メモリ上の所与の第一のエリアから所与の第一のレジスタへのデータの転送及び前記所与の第一のレジスタから前記所与の第一のエリアへのデータの転送の少なくとも一方を、前記スタックポインタ専用レジスタにより特定されるメモリアドレス及び前記転送レジスタ特定情報により特定されるレジスタアドレスに基づき行うことを特徴とする情報処理回路。

【請求項3】　請求項2において、

前記ロード命令が、前記メモリ上の前記第一のエリアのアドレスを特定するためのオフセットに関する情報であるオフセット情報をオブジェクトコードに含み、

前記実行手段が、

前記スタックポインタ専用レジスタの内容と前記オフセット情報により前記メモリアドレスを特定することを特徴とする情報処理回路。

【請求項4】　請求項3において、

前記オフセット情報が、即値で与えられた即値オフセット情報とメモリ上の所与のデータのサイズに関するデータサイズ情報とを含み、

前記実行手段が、

前記即値オフセット情報と前記データサイズ情報とに基づき、前記即値オフセット情報を左論理シフトしてオフセット値を生成し、前記スタックポインタ専用レジスタの内容と前記オフセット値を加算した値により前記メモリアドレスを特定することを特徴とする情報処理回路。

【請求項5】　請求項1～請求項4のいずれかにおいて、

前記スタックポインタ専用命令群が、オブジェクトコードに移動情報を有しスタックポインタを移動するためのスタックポインタ移動命令を含み、

前記解読手段が、

前記スタックポインタ移動命令を解読し、

前記実行手段が、

2

前記スタックポインタ移動命令を実行する際、前記移動情報に基づき、前記スタックポインタ専用レジスタの内容を変更することを特徴とする情報処理回路。

【請求項6】　請求項5において、

前記移動情報が、即値で与えられた即値移動情報を含み、

前記命令実行手段が、

前記即値移動情報と前記スタックポインタ専用レジスタの内容とを加算する処理及び前記スタックポインタ専用レジスタの内容から前記即値移動情報を減算する処理の少なくとも一方の処理を行うことを特徴とする情報処理回路。

【請求項7】　請求項1～請求項6のいずれかにおいて、

連続して順序づけられた複数のレジスタを含み、

前記スタックポインタ専用命令群が、複数レジスタ特定情報をオブジェクトコードに有する連続プッシュ命令及び連続ポップ命令の少なくとも一方を含み、

前記解読手段が、

前記連続プッシュ命令及び連続ポップ命令の少なくとも一方を解読し、

前記命令実行手段が、

前記連続プッシュ命令及び前記連続ポップ命令の少なくとも一方を実行する際、前記複数のレジスタからメモリに設けられたスタックへ連続して複数回プッシュする処理及び前記スタックから前記複数のレジスタに連続して複数回ポップする処理の少なくとも一方を、前記スタックポインタ専用レジスタの内容により特定されるメモリアドレスに及び前記複数レジスタ特定情報とに基づき行うことを特徴とする情報処理回路。

【請求項8】　請求項7において、

0からn－1までのレジスタ番号で特定されたn個の汎用レジスタを含み、

前記連続プッシュ命令及び連続ポップ命令の少なくとも一方のオブジェクトコードが、前記複数レジスタ特定情報として、前記レジスタ番号のいずれかが指定された最終レジスタ番号を含み、

前記実行手段が、

レジスタ0から前記最終レジスタ番号で特定されるレジスタまでの複数のレジスタからメモリに設けられたスタックへ連続して複数回プッシュする処理及び前記スタックから前記複数のレジスタに連続して複数回ポップする処理の少なくとも一方を、前記スタックポインタ専用レジスタの内容により特定されるメモリアドレスに基づき行うことを特徴とする情報処理回路。

【請求項9】　請求項7又は8のいずれかにおいて、

前記実行手段が、

前記複数のレジスタのいずれか所与のレジスタの内容を、前記スタックポインタ専用レジスタで特定されるメモリアドレスに基づきメモリに設けられたスタックに書

き込む書き込み手段と、
前記書き込み手段による前記スタックへの書き込み回数をカウントする書き込み回数カウント手段と、
前記カウント手段によってカウントされた前記書き込み回数と前記複数レジスタ特定情報の値を比較する比較手段とを含み、
前記書き込み手段が、
第一の入力と第二の入力を加算器で加算し書き込み先を特定するための書き込みメモリアドレスを生成する書き込みメモリアドレス生成手段と、
前記加算器の第一の入力が、連続専用命令の実行開始時にはスタックポインタ専用レジスタの内容となるように制御し、それ以降は書き込みアドレス生成手段によって生成された書き込みアドレスとなるよう制御する第一の入力制御手段と、
前記加算器の第二の入力に前記スタックから１ワードを書き込んだときのオフセット値を出力する第二の入力制御手段と、
前記複数レジスタ特定情報から前記書き込み回数を減じた値で特定されるレジスタの内容を、前記書き込みメモリアドレスに基づき前記スタックに書き込む手段とを含み、
前記比較手段の比較結果に基づき、複数のレジスタからの前記スタックへの書き込み及び書き込み終了を制御することを特徴とする情報処理回路。
【請求項１０】　請求項７～９のいずれかにおいて、
前記命令実行手段が、
前記スタックポインタ専用レジスタで特定されるメモリアドレスに基づきメモリに設けられたスタックの内容を読み出し、前記複数のレジスタのいずれか所与のレジスタに格納する読み出し手段と、
前記読み出し手段による前記スタックからの読み出し回数をカウントする読み出し回数カウント手段と、
前記カウント手段によってカウントされた前記読み出し回数と前記複数レジスタ特定情報の値を比較する比較手段とを含み、
前記読み出し手段が、
第一の入力と第二の入力を加算器で加算し書き込み先を特定するための書き込みメモリアドレスを生成する書き込みメモリアドレス生成手段と、
前記加算器の第一の入力が、連続専用命令の実行開始時にはスタックポインタ専用レジスタの内容となるように制御し、それ以降は読み出しアドレス生成手段によって生成された読み出しアドレスとなるよう制御する第一の入力制御手段と、
前記加算器の第二の入力に前記スタックから１ワードを書き込んだときのオフセット値を出力する第二の入力制御手段と、
前記読み出しメモリアドレスに基づき前記スタックの内容を読み出し、前記書き込み回数に基づき特定されるレジスタに格納する読み出し手段とを含み、
前記比較手段の比較結果に基づき、前記スタックの内容の読み出し及び読み出し終了を制御することを特徴とする情報処理回路。
【請求項１１】　請求項１～請求項１０のいずれかにおいて、
プログラムカウンタ専用のプログラムカウンタレジスタを含み、
前記スタックポインタ専用命令群が、サブルーチンへ分岐する命令及び前記サブルーチンからのリターン命令である分岐命令を含み、
前記解読手段が、
前記分岐命令を解読し、
前記命令実行手段が、
前記分岐命令を実行する際、メモリに設けられたスタックの所与の第二のエリアへの前記プログラムカウンタレジスタの内容の待避及び前記第二のエリアの内容のプログラムカウンタレジスタへの復帰の少なくとも一方を、前記スタックポインタ専用レジスタにより特定されるメモリアドレスに基づき行う手段と、
前記待避及び前記復帰に基づき前記スタックポインタ専用レジスタの内容を更新する手段とを含むことを特徴とする情報処理回路。
【請求項１２】　連続して順序づけられた複数のレジスタと、いずれかの汎用レジスタに割り当てられたスタックポインタを含む情報処理回路であって、
複数レジスタ特定情報をオブジェクトコードに有する連続プッシュ命令及び連続ポップ命令の少なくとも一方の命令のオブジェクトコードを解読して、該オブジェクトコードに基づき制御信号を出力する手段と、
前記連続プッシュ命令及び前記連続ポップ命令の少なくとも一方を実行する際、前記複数のレジスタからメモリに設けられたスタックへ連続して複数回プッシュする処理及び前記スタックから前記複数のレジスタに連続して複数回ポップする処理の少なくとも一方を、前記制御信号及び前記スタックポインタ専用レジスタの内容により特定されるメモリアドレス及び前記複数レジスタ特定情報とに基づき行う手段を含むことを特徴とする情報処理回路。
【請求項１３】　請求項１～請求項１２のいずれかにおいて、ＲＩＳＣ方式であることを特徴とする情報処理回路。
【請求項１４】　請求項１～請求項１３のいずれかにおいて、
固定長の命令を解読し、該命令に基づき実行処理を行うことを特徴とする情報処理回路。
【請求項１５】　請求項１～請求項１４のいずれかに記載の情報処理回路と記憶手段と外部との入出力を行う手段とを含むことを特徴とするマイクロコンピュータ。
【請求項１６】　請求項１５において、

前記スタックポインタに関連づけてオート変数の記憶領域が確保される構造を有する言語のプログラムが実行されることを特徴とするマイクロコンピュータ。

【請求項17】 請求項15又は16のいずれかに記載されたマイクロコンピュータを含むことを特徴とする電子機器。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】本発明は、情報処理回路、前記情報処理回路を内蔵するマイクロコンピュータ、及び該マイクロコンピュータを用いて構成された電子機器に関する。

【0002】

【背景技術及び発明が解決しようとする課題】従来、32ビットのデータを処理できるRISC方式のマイクロコンピュータでは、32ビット幅の固定長命令が用いられていた。その理由は、固定長命令を用いると、可変長命令を用いる場合に比べ、命令のデコードに要する時間を短縮でき、また、マイクロコンピュータの回路規模を小さくすることが出来るからである。

【0003】ところが、32ビットのマイクロコンピュータにおいても、特に32ビットも必要としない命令も多い。従って全ての命令を32ビットで記述すると、命令に冗長な部分が生じる命令が多くなり、メモリーの使用効率が悪くなる。

【0004】そこで、本願の発明者は、制御回路を複雑にすることなくメモリの使用効率を向上させるために、処理出来るデータのビット幅より短いビット幅の固定長命令を処理するマイクロコンピュータについての検討を行っていた。

【0005】しかし、例えば32ビット固定長の命令を単に16ビットの固定長にすると以下のような問題点が生じる。

【0006】即ちRISC方式のマイクロコンピュータでは処理及び命令セットの汎用性を重視するため、スタックポインタを取り扱う処理を行う場合、いずれかの汎用レジスタをスタックポインタとして用い、汎用レジスタを操作対象とする命令を用いて前記処理を行っていた。従って、このような処理を記述する際の前記命令のには、スタックポインタとして使用している汎用レジスタの指定が必要となる。

【0007】例えば、スタックポインタに所与のオフセット値を加えたメモリアドレスで特定されるデータを所与のレジスタに転送する処理を汎用レジスタを操作対象とする命令で記述する場合、その命令のオブジェクトコードには、前記オフセット値、前記所与のレジスタを特定する情報、スタックポインタとして使用しているレジスタを特定する情報が必要となる。

【0008】このように、汎用レジスタを操作対象とする命令でスタックポインタを取り扱う処理を記述する場合、オブジェクトコードで指定する情報が多くなるため、命令の内容を16ビットの固定長で記述することが困難となる。ここで命令長を例えば32ビットにすると、命令の中にも特に命令に32ビットも必要としない命令も多いため、命令に冗長な部分を生じる命令が多くなり、メモリの使用効率の悪化を招く。

【0009】また、命令長が長くなればそれを格納するメモリも余分に必要になるため、メモリの効率的な使用という観点からは、固定長の命令に限らず、命令長を短くできることが好ましい。

【0010】また、例えばC言語のようにスタックポインタに関連づけてオート変数の記憶領域が確保される言語で記述されたプログラムを実行させる場合、スタックポインタを取り扱う命令が多くなるため、スタックポインタを取り扱う命令を効率的に記述し、実行させることが望ましい。

【0011】そこで、スタックポインタを取り扱う処理を行う場合、できるだけ短い命令長でその命令内容を記述し、実行することができるアーキテクチャが望まれていた。

【0012】また最近の特にRISC方式のCPUは、性能を高めるため内部に多くの汎用レジスタを持つようになった。多くのレジスタを内部に持つことによりメモリにアクセスすることなくCPU内部で高速に多くの処理を出来るからである。このように内部レジスタを多く持つと、割り込み処理やサブルーチンコールの時のレジスタ退避と復旧の処理の際、退避すべきレジスタ数が多くなる。

【0013】以下、スタック系命令の中でもサブルーチンに入ったりでたりする際に多用するレジスタの待避、復帰命令を例にとり従来例を説明する。

【0014】通常マイクロコンピュータの命令セットはCPUのレジスタをメモリに設けたスタックに待避したり復旧するための命令を持っている。そのための専用の命令を持つもの、あるいはレジスタ間接アドレッシング命令をもつものがある。

【0015】前記待避したり復旧したりするための命令に関する技術としては、インテル社の80386に関して、「80386プログラミン」（John H. Crawford Patric P. Gelsigner 著 岩谷宏 訳）に以下のような記述がある。

【0016】即ちレジスタをスタックに書き込む命令として push、pusha、pushadがあり，スタックからレジスタにデータを戻すための命令としてpop、popa、popadがある。

【0017】push命令でレジスタをスタックに書き込むときは push EAX のようにオペランドとしてレジスタを指定する。これは32ビットレジスタEAXの場合である。レジスタEAX、ECX、EDX、EBXを全てスタックに書く場合は、

ｐｕｓｈ　ＥＡＸ
ｐｕｓｈ　ＥＣＸ
ｐｕｓｈ　ＥＤＸ
ｐｕｓｈ　ＥＢＸ
のようにｐｕｓｈ命令を繰り返す。

【００１８】このようにｐｕｓｈ、ｐｏｐ命令でレジスタを１本ずつ操作しているとオブジェクトコードのサイズは大きくなり、プログラムの実行ステップも多くなり、そのためプログラムの実行時間や処理動作はより遅くなる。

【００１９】そこで８０３８６の有する８本の汎用レジスタ全てをスタックに書き込むにはｐｕｓｈａ　またはｐｕｓｈａｄ　命令を使う。ｐｕｓｈａは８本のレジスタの其々の下位１６ビットレジスタを対象とし、ｐｕｓｈａｄは３２ビットを対象とする。ｐｕｓｈａ、ｐｕｓｈａｄによりｐｕｓｈ命令を８回繰り返すことを省略できる。

【００２０】ｐｏｐ、ｐｏｐａ、ｐｏｐａｄ命令についても同様である。

【００２１】ｐｕｓｈ命令を繰り返すことの不利な点はプログラムコードが長くなること、１命令毎フェッチし実行するため実行が遅いことである。この点は、前記ｐｕｓｈａ、ｐｕｓｈａｄにより、前記８本のレジスタを全てスタックに書き込む場合には大きく改善される。しかし、４本の場合とか６本の場合等のように８未満のレジスタを書き込む場合には、この点は改善されない。

【００２２】即ち、８０３８６のようにｐｕｓｈａ、ｐｕｓｈａｄ、ｐｏｐａ、ｐｏｐａｄ命令で全てのレジスタを操作すると、レジスタの全てを退避・復旧する必要の無いときも、このサイクルの長い遅い命令を使わなければならない。このような場合、命令は一命令で済むがこの命令の実行サイクルは長くなるという問題が生じる。

【００２３】また、ｃａｌｌ命令やリターン命令等でサブルーチンへ分岐したり、呼ばれたルーチンへ戻る際には、戻り先アドレスとして必要となるプログラムカウンタの待避や復旧の処理が必要となる。従来のＲＩＳＣ方式のＣＰＵにおいては、これらの処理をソフトウエア的に実現していた。即ち、該処理を記述したアセンブラ命令（オブジェクトコード）を実行することにより、前記プログラムカウンタの待避や復旧の処理を行っていた。このため、ｃａｌｌ命令やリターン命令はオブジェクトコードの増加を招き、また１命令毎フェッチし実行するために実行速度の鈍化を招いていた。

【００２４】本発明の目的は、スタックポインタを取り扱う処理を、短い命令長で効率よく記述し、実行することができるアーキテクチャを有する情報処理回路、マイクロコンピュータ、電子機器を提供することである。

【００２５】また　本発明の他の目的は、レジスタ退避やレジスタ復旧の処理を、効率よく記述し、割り込み処理及びサブルーチンコール・リターンの処理速度が速い情報処理回路、マイクロコンピュータ、電子機器を提供することである。

【００２６】

【課題を解決するための手段】本発明は、スタックポインタ専用に用いるスタックポインタ専用レジスタと、該スタックポインタ専用レジスタを暗黙のオペランドとするオブジェクトコードを有し、該スタックポインタ専用レジスタに基づく処理が記述されたスタックポインタ専用命令群のオブジェクトコードを解読して、該オブジェクトコードに基づき制御信号を出力する解読手段と、前記スタックポインタ専用命令群を、前記制御信号及び前記スタックポインタ専用レジスタの内容に基づき実行する実行手段とを含むことを特徴とする。

【００２７】ここにおいてオブジェクトコードとは、一般に翻訳プログラムによって機械語に翻訳した結果得られるプログラムコードのことをいうが、本発明においては、翻訳プログラムによるかいなかに関係なく、機械語で記述されているプログラムコードを含む広い概念で使用する。

【００２８】本発明の情報処理回路は、スタックポインタ専用のスタックポインタ専用レジスタを有し、該スタックポインタ専用レジスタを操作対象とするスタックポインタ専用命令群の解読、実行を行うよう構成されている。

【００２９】前記スタックポインタ専用命令群は、スタックポインタ専用レジスタを取り扱うための専用のオペコードを有しているため、オブジェクトコードのオペランドにスタックポインタを特定するための情報を必要としない。言い換えれば、前記スタックポインタ専用命令群はスタックポインタ専用レジスタを暗黙のオペランドとしている。このため、汎用レジスタの１つをスタックポインタに割あて、汎用レジスタを操作対象とする命令を用いてスタックポインタの操作を行う場合に比べて、短い命令長でスタックポインタを取り扱う命令を記述することができる。

【００３０】従って本発明によれば、スタックポインタを取り扱う処理を短い命令長で記述し実行することができる情報処理回路を提供することができる。また、命令を記憶するメモリの使用効率のよい情報処理回路を提供することができる。

【００３１】本発明は、前記スタックポインタ専用命令群が、転送レジスタ特定情報をオブジェクトコードに有するロード命令を含み、前記解読手段が、前記ロード命令を解読し、前記実行手段が、前記ロード命令を実行する際、メモリ上の所与の第一のエリアから所与の第一のレジスタへのデータの転送及び前記所与の第一のレジスタから前記所与の第一のエリアへのデータの転送の少なくとも一方を、前記スタックポインタ専用レジスタにより特定されるメモリアドレス及び前記転送レジスタ特定

情報により特定されるレジスタアドレスに基づき行うことを特徴とする。

【００３２】ここにおいて前記スタックポインタ専用命令群に含まれる前記ロード命令とは、メモリとレジスタ間でデータの転送を行う命令であり、メモリからレジスタへの転送及びレジスタからメモリへの転送の少なくとも一方を含む概念である。なお、データの内容は問わずアドレスデータも含む概念である。メモリアドレスとは、転送の際のメモリ上のエリアを特定するためのアドレスをいう。

【００３３】前記ロード命令は、スタックポインタ専用レジスタを操作対象とするための専用のオペコードを有しているため、オブジェクトコードのオペランドにスタックポインタを特定するための情報を必要としない。このため、スタックポインタに関連づけたメモリアドレスを有するメモリ上のエリアとレジスタ間でデータの転送処理を行わせる場合、短い命令長で記述することができる。

【００３４】本発明は、前記ロード命令が、前記メモリ上の前記第一のエリアのアドレスを特定するためのオフセットに関する情報であるオフセット情報をオブジェクトコードに含み、前記実行手段が、前記スタックポインタ専用レジスタの内容と前記オフセット情報により前記メモリアドレスを特定することを特徴とする。

【００３５】ここにおいてオフセット情報とは、オフセット値を直接即値で指定したものでもよいし、オフセット値が格納されたレジスタ等のアドレスを指定する場合のように間接的に指定する場合でもよい。オフセット情報を含んだ前記ロード命令が実行される場合、転送の際に必要となる前記メモリアドレスは、スタックポインタ専用レジスタの内容及びオフセット情報に基づき特定される。

【００３６】従ってスタックポインタ及び前記オフセット情報に基づき特定されるメモリアドレスを有するメモリ上のエリアとレジスタ間でデータの転送処理を行わせる場合、短い命令長で記述することができる。

【００３７】また本発明によれば、例えばスタックポインタが常にワード境界をさすような構造の情報処理回路においても、適当なオフセット情報を指定することにより、スタック上の任意のエリアを指定することが可能となる。このため、データのサイズに応じて効率良くスタックに格納することができ、スタックの使用効率の向上を図ることができる。

【００３８】本発明は、前記オフセット情報が、即値で与えられた即値オフセット情報とメモリ上の所与のデータのサイズに関するデータサイズ情報とを含み、前記実行手段が、前記即値オフセット情報と前記データサイズ情報とに基づき、前記即値オフセット情報を左論理シフトしてオフセット値を生成し、前記スタックポインタ専用レジスタの内容と前記オフセット値を加算した値により前記メモリアドレスを特定することを特徴とずる。

【００３９】ここにおいて即値オフセット情報とは、オフセット値を直接即値で指定したものをいう。またデータサイズ情報とは、転送すべきメモリ上データのサイズをいう。通常データサイズは、８ビットのバイトデータや、１６ビットのハーフワードデータ、３２ビットのワードデータ等の2^n（nは3以上）で表される。メモリ上のアドレスはバイト単位に与えられており、ハーフワードデータはハーフワード境界におかれ、ワードデータはワード境界におかれる。従って、ハーフワードデータのメモリアドレスの下位の１ビットは０となり、ワードデータのメモリアドレスの下位の２ビットは００となる。スタックポインタのアドレスは、ワード境界を指しているため、ハーフワードデータのメモリアドレスを生成する場合のオフセット値の下位の１ビットは０となり、ワードデータのメモリアドレスを生成する場合のオフセット値の下位の２ビットは００となる。

【００４０】また、左論理シフトとは、データのビット列を左にずらし、シフトによってデータの右側にでるあきビット（シフトインビット）に０が入るシフトをいう。

【００４１】本発明によれば、データサイズに基づき、前記オフセット即値情報の左論理シフトを行うため、データサイズによって一義的に決まる下位のビットを省略して、即値オフセット情報を記述することができる。従って、即値オフセット情報を効率的に指定することができ、バイト以外のデータサイズの場合、そのまま指定する場合に比べて、より大きいオフセット値に指定が可能となる。

【００４２】また該命令を用いることによって、データのメモリへの書き込み、読み出し時にそのデータサイズに応じた適切な境界位置が選択されることになる。

【００４３】本発明は、前記スタックポインタ専用命令群が、オブジェクトコードに移動情報を有しスタックポインタを移動するためのスタックポインタ移動命令を含み、前記解読手段が、前記スタックポインタ移動命令を解読し、前記実行手段が、前記スタックポインタ移動命令を実行する際、前記移動情報に基づき、前記スタックポインタ専用レジスタの内容を変更することを特徴とする。

【００４４】前記スタックポインタ移動命令は、スタックポインタ専用レジスタを操作対象とするための専用のオペコードを有しているため、オブジェクトコードのオペランドにスタックポインタを特定するための情報を必要としない。このため、スタックを移動させたい場合、短い命令長で記述することができる。従って、スタックに格納されているデータの処理やスタックポインタに関連づけて記憶されているデータの処理を行う際の命令の記述量を削減することができる。

【００４５】本発明によればスタックの移動が容易に行

えるため、特に異なるルーチンにおいてそれぞれ異なるスタックエリアを確保して処理を行う場合に有効である。即ちルーチンごとにスタックポインタを移動させる処理を行うことで、広範囲な領域にわたるアドレス指定が可能となる。

【００４６】本発明は、前記移動情報が、即値で与えられた即値移動情報を含み、前記命令実行手段が、前記即値移動情報と前記スタックポインタ専用レジスタの内容とを加算する処理及び前記スタックポインタ専用レジスタの内容から前記即値移動情報を減算する処理の少なくとも一方の処理を行うことを特徴とする。

【００４７】本発明によれば、スタックポインタの値を即値移動情報で指定された分だけ上方又は下方に移動させる処理を短い命令長で記述することができる。

【００４８】本発明は、連続して順序づけられた複数のレジスタを含み、前記スタックポインタ専用命令群が、複数レジスタ特定情報をオブジェクトコードに有する連続プッシュ命令及び連続ポップ命令の少なくとも一方を含み、前記解読手段が、前記連続プッシュ命令及び連続ポップ命令の少なくとも一方を解読し、前記命令実行手段が、前記連続プッシュ命令及び前記連続ポップ命令の少なくとも一方を実行する際、前記複数のレジスタからメモリに設けられたスタックへ連続して複数回プッシュする処理及び前記スタックから前記複数のレジスタに連続して複数回ポップする処理の少なくとも一方を、前記スタックポインタ専用レジスタの内容により特定されるメモリアドレスに及び前記複数レジスタ特定情報とに基づき行うことを特徴とする。

【００４９】プッシュとはメモリに設けられたスタックにデータを積み重ねて格納すること、ポップとは前記スタックからデータを取り出すことをいう。ここにおいてプッシュする処理及びポップする処理とは、前記格納及び取り出しの処理と、それに伴うスタックポインタの更新処理を含む。通常の情報処理回路は１のレジスタからスタックにデータやアドレスを格納するためのｐｕｓｈ命令、スタックの内容をレジスタに取り出すｐｏｐ命令を有している。該ｐｕｓｈ命令やｐｏｐ命令は、レジスタとスタックとのデータのやり取りや、該やり取りに伴うスタックポインタの更新を行う。

【００５０】従って複数のレジスタとスタックでデータ等のやり取りを行う場合には、これらの命令を複数回実行することが必要となる。

【００５１】しかし本発明によれば、前記連続プッシュ命令や前記連続ポップ命令を実行すると、ｐｕｓｈ命令を連続して複数回実行する場合やｐｏｐ命令を連続して複数回実行する場合と同じ効果が得られる。即ち複数のレジスタとスタックとの間のデータのやり取り及び該やり取りに伴うスタックポインタの更新を１命令で実行することが出来る。このため複数のレジスタとスタックとの間のデータの転送を行う場合、ｐｕｓｈ命令あるいはｐｏｐ命令をくり返すことによりオブジェクトコードサイズが増大するのを防ぐことができる。またプログラム実行ステップが長くなることを回避し、無駄なサイクルを消費することなく、割り込み処理及びサブルーチンコール・リターンの処理速度の向上を図ることができる。

【００５２】本発明は、０からｎ－１までのレジスタ番号で特定されたｎ個の汎用レジスタを含み、前記連続プッシュ命令及び連続ポップ命令の少なくとも一方のオブジェクトコードが、前記複数レジスタ特定情報として、前記レジスタ番号のいずれかが指定された最終レジスタ番号を含み、前記実行手段が、レジスタ０から前記最終レジスタ番号で特定されるレジスタまでの複数のレジスタからメモリに設けられたスタックへ連続して複数回プッシュする処理及び前記スタックから前記複数のレジスタに連続して複数回ポップする処理の少なくとも一方を、前記スタックポインタ専用レジスタの内容により特定されるメモリアドレスに基づき行うことを特徴とする。

【００５３】通常汎用レジスタが複数ある場合、レジスタを特定するためのアドレスを有している。本発明では０からｎ－１までの連続したレジスタ番号で前記レジスタを特定している。

【００５４】本発明によれば、前記最終レジスタ番号に任意のレジスタ番号を指定することにより、レジスタ０から前記最終レジスタ番号までの連続した複数のレジスタとメモリとの間でデータをプッシュする処理及びポップする処理の少なくとも一方が行われる。従って、レジスタ番号０のレジスタから順番にレジスタを使用するような構造を有するプログラムの実行において、レジスタの待避や復旧を効率的に行うことができる。

【００５５】本発明は、前記実行手段が、前記複数のレジスタのいずれか所与のレジスタの内容を、前記スタックポインタ専用レジスタで特定されるメモリアドレスに基づきメモリに設けられたスタックに書き込む書き込み手段と、前記書き込み手段による前記スタックへの書き込み回数をカウントする書き込み回数カウント手段と、前記カウント手段によってカウントされた前記書き込み回数と前記複数レジスタ特定情報の値を比較する比較手段とを含み、前記書き込み手段が、第一の入力と第二の入力を加算器で加算し書き込み先を特定するための書き込みメモリアドレスを生成する書き込みメモリアドレス生成手段と、前記加算器の第一の入力が、連続専用命令の実行開始時にはスタックポインタ専用レジスタの内容となるように制御し、それ以降は書き込みアドレス生成手段によって生成された書き込みアドレスとなるよう制御する第一の入力制御手段と、前記加算器の第二の入力に前記スタックから１ワードを書き込んだときのオフセット値を出力する第二の入力制御手段と、前記複数レジスタ特定情報から前記書き込み回数を減じた値で特定されるレジスタの内容を、前記書き込みメモリアドレスに

基づき前記スタックに書き込む手段とを含み、前記比較手段の比較結果に基づき、複数のレジスタからの前記スタックへの書き込み及び書き込み終了を制御することを特徴とする。

【００５６】また本発明は、前記命令実行手段が、前記スタックポインタ専用レジスタで特定されるメモリアドレスに基づきメモリに設けられたスタックの内容を読み出し、前記複数のレジスタのいずれか所与のレジスタに格納する読み出し手段と、前記読み出し手段による前記スタックからの読み出し回数をカウントする読み出し回数カウント手段と、前記カウント手段によってカウントされた前記読み出し回数と前記複数レジスタ特定情報の値を比較する比較手段とを含み、前記読み出し手段が、第一の入力と第二の入力を加算器で加算し書き込み先を特定するための書き込みメモリアドレスを生成する書き込みメモリアドレス生成手段と、前記加算器の第一の入力が、連続専用命令の実行開始時にはスタックポインタ専用レジスタの内容となるように制御し、それ以降は読み出しアドレス生成手段によって生成された読み出しアドレスとなるよう制御する第一の入力制御手段と、前記加算器の第二の入力に前記スタックから１ワードを書き込んだときのオフセット値を出力する第二の入力制御手段と、前記読み出しメモリアドレスに基づき前記スタックの内容を読み出し、前記書き込み回数に基づき特定されるレジスタに格納する読み出し手段とを含み、前記比較手段の比較結果に基づき、前記スタックの内容の読み出し及び読み出し終了を制御することを特徴とする。

【００５７】このようにすると連続した値で順序づけて特定される複数のレジスタのスタックへの待避又はスタックから連続した値で特定される複数のレジスタへの復旧をカウント手段と簡単なシーケンス制御のみで実現可能である。従って少ないゲート数の情報処理回路で実現出来るため、ワンチップのマイクロコンピュータ等に適したものである。

【００５８】本発明は、プログラムカウンタ専用のプログラムカウンタレジスタを含み、前記スタックポインタ専用命令群が、サブルーチンへ分岐する命令及び前記サブルーチンからのリターン命令である分岐命令を含み、前記解読手段が、前記分岐命令を解読し、前記命令実行手段が、前記分岐命令を実行する際、メモリに設けられたスタックの所与の第二のエリアへの前記プログラムカウンタレジスタの内容の待避及び前記第二のエリアの内容のプログラムカウンタレジスタへの復帰の少なくとも一方を、前記スタックポインタ専用レジスタにより特定されるメモリアドレスに基づき行う手段と、前記待避及び前記復帰に基づき前記スタックポインタ専用レジスタの内容を更新する手段とを含むことを特徴とする。

【００５９】ここにおいてサブルーチンとは、割り込み処理及び例外処理及びデバッグ処理ルーチン等も含む。従ってサブルーチンへ分岐する命令とは、サブルーチン等をコールする命令、割り込み処理及び例外処理及びデバッグ処理ルーチン等へ分岐するためのソフトウェア割り込み命令、ソフトウェアデバッグ割り込み命令等も含む。またサブルーチンからのリターン命令には、割り込み処理及び例外処理及びデバッグ処理ルーチン等からのリターン命令を含む。

【００６０】通常サブルーチンへ分岐する場合や前記サブルーチンから戻る場合プログラムカウンタの待避や復帰が必要となる。

【００６１】本発明では、前記サブルーチンへ分岐する命令の実行及びサブルーチンからリターンする命令を実行する際に前記プログラムカウンタの待避及び復帰も同時に行う。即ち、本発明の情報処理回路は、サブルーチンへ分岐する命令及び前記サブルーチンからのリターン命令のいずれか一命令でプログラムカウンタの待避及び復帰が出来る回路構成を有している。このためサブルーチンへの分岐やサブルーチンからのリターンに伴って必要となる前記プログラムカウンタの待避及び復帰の命令が不要となり、命令数を削減することができる。また、無駄なサイクルを消費することなく、サブルーチンコール・リターン等の他のルーチンへ分岐時の処理速度の向上を図ることができる。

【００６２】また、ソフトウエア割り込み命令が発生した場合には、例えばＣＰＵ等の情報処理回路の現在の状態を保持するプロセッサーステータスレジスタの待避及び復帰も必要となる。従って、ソフトウエア割り込み命令等の場合は該命令の実行時にプロセッサーステータスレジスタの待避及び復帰も同時に行うようにすることが好ましい。

【００６３】本発明は、連続して順序づけられた複数のレジスタと、いずれかの汎用レジスタに割り当てられたスタックポインタを含む情報処理回路であって、複数レジスタ特定情報をオブジェクトコードに有する連続プッシュ命令及び連続ポップ命令の少なくとも一方の命令のオブジェクトコードを解読して、該オブジェクトコードに基づき制御信号を出力する手段と、前記連続プッシュ命令及び前記連続ポップ命令の少なくとも一方を実行する際、前記複数のレジスタからメモリに設けられたスタックへ連続して複数回プッシュする処理及び前記スタックから前記複数のレジスタに連続して複数回ポップする処理の少なくとも一方を、前記制御信号及び前記スタックポインタ専用レジスタの内容により特定されるメモリアドレス及び前記複数レジスタ特定情報とに基づき行う手段を含むことを特徴とする。

【００６４】本発明は、汎用レジスタをスタックポインタとして使用する場合の前記連続プッシュ命令及び前記連続ポップ命令に関する。

【００６５】本発明によれば、前記連続プッシュ命令や前記連続ポップ命令を実行すると、ｐｕｓｈ命令を連続して複数回実行する場合やｐｏｐ命令を連続して複数回

実行する場合と同じ効果が得られる。即ち複数のレジスタとスタックとの間のデータのやり取り及び該やり取りに伴うスタックポインタの更新を１命令で実行することが出来る。このため複数のレジスタとスタックとの間のデータの転送を行う場合、ｐｕｓｈ命令あるいはｐｏｐ命令をくり返すことによりオブジェクトコードサイズが増大するのを防ぐことができる。またプログラム実行ステップが長くなることを回避し、無駄なサイクルを消費することなく、割り込み処理及びサブルーチンコール・リターンの処理速度の向上を図ることができる。

【００６６】本発明はの情報処理回路はＲＩＳＣ方式であることを特徴とする。

【００６７】ＲＩＳＣ方式の情報処理回路は、ハードウエアを小型化して高速化を図ることを目的として設計されている。このため汎用レジスタを多く有しており、命令セットを汎用性の高いものに絞ることで命令数の削減を図っている。

【００６８】従ってＲＩＳＣ方式の情報処理回路では、スタックポインタを汎用レジスタに割り当て、スタックポインタを扱う場合は汎用レジスタを扱う命令セットを用いて処理を行っていた。しかしこの様な方法では命令長が大きくなりメモリの使用効率がよくない。

【００６９】本発明によれば、ＲＩＳＣ方式の情報処理回路において、命令長を削減することが出来、メモリの使用効率を上げることが出来る。

【００７０】本発明の情報処理回路は、固定長の命令を解読し、該命令に基づき実行処理を行うことを特徴とする。

【００７１】固定長命令を用いると可変長命令を用いる場合に比べ、命令のデコードに要する時間を短縮でき、情報処理回路の回路規模を小さくすることが出来る。固定長命令を採用する場合、命令に冗長な部分が出来るのを防ぎメモリを効率よく使用するためには、各命令に必要なビット数はばらつきが少なく、出来るだけ短いほうが好ましい。

【００７２】本発明によれば、一般に命令長が長くなりがちなスタックポインタを取り扱う命令の命令長を短くすることができる。従って固定長命令を採用した場合であっても命令に冗長な部分が出来るのを防ぎ、メモリを効率よく使用することが出来る。

【００７３】本発明のマイクロコンピュータは、前述した本発明の情報処理回路と記憶手段と外部との入出力を行う手段とを含むことを特徴とする。

【００７４】本発明によれば、処理速度が速くメモリの使用効率のよい、マイクロコンピュータを提供することが出来る。

【００７５】本発明のマイクロコンピュータは、前記スタックポインタに関連づけてオート変数の記憶領域が確保される構造を有する言語のプログラムが実行されることを特徴とする。

【００７６】スタックポインタに関連づけてオート変数の記憶領域が確保される構造を有するする言語として、例えばＣ言語がある。この様な言語のプログラムの処理を本発明のマイクロコンピュータが行う場合、処理速度及びメモリの使用効率を効果的に向上させることが出来る。

【００７７】本発明の電子機器は、前述した本発明のマイクロコンピュータを含むことを特徴とする。

【００７８】本発明によれば、処理速度が速くメモリの使用効率のよい情報処理回路を内蔵しているため、安価で高機能な電子機器を提供することが出来る。

【００７９】

【発明の実施の形態】以下、本実施例を図面に基づき説明する。

【００８０】（実施例１）

（１）本実施例のＣＰＵの構成

本実施例のＣＰＵはパイプラインとロード・ストア型のアーキテクチュアによって、ほとんど全ての命令を１サイクルで実行する。全ての命令は１６ビットの固定長で記述されており、本実施例のＣＰＵの処理する命令は極めて小さいオブジェクトコードサイズを実現している。

【００８１】特に本実施例のＣＰＵは、スタックポインタを取り扱う処理を効率よく記述し実行するためにスタックポインタ専用のレジスタを有し、該スタックポインタ専用レジスタを暗黙のオペランドとするオブジェクトコードを有するスタックポインタ専用命令群の命令セットを解読、実行出来るよう構成されている。

【００８２】図１は、本実施例のＣＰＵの回路構成の概略を説明するための図である。

【００８３】本ＣＰＵ１０は、汎用レジスタ１１、プログラムカウンタが格納されているＰＣ１２、プロセッサーステータスレジスタ（ＰＳＲ）１３、スタックポインタ専用のレジスタであるＳＰ１４を含むレジスタセットと、命令デコーダ２０、即値生成器２２、アドレス加算器３０、ＡＬＵ４０、ＰＣインクリメンタ４４及び各種内部バス７２、７４、７６、７８、各種内部信号線８２、８４、８６、８８等を含む。

【００８４】前記命令デコーダ２０は、入力したオブジェクトコードを解読し、該命令を実行するために必要な処理を行い、必要な制御信号を出力する。なお、該命令デコーダ２０は前記スタックポインタ専用命令のオブジェクトコードを解読して該命令に基づき制御信号を出力する前記解読手段としても機能する。

【００８５】即値生成器２２は、オブジェクトコードに含まれた即値に基づき、実行時に使用する３２ビットの即値データを生成したり、各命令の実行に必要な０、±１、±２、±４のconstantデータを生成したりする。ＰＣインクリメンタ４４は、命令の実行サイクルに基づきＰＣ１２に格納されたプログラムカウンタの更新を行う。アドレス加算器３０は、各種レジスタに格納されて

いる情報や即値生成器２２で生成される即値データの加算を行いメモリからデータを読み出す際に必要なアドレスデータを生成する。ＡＬＵ４０は数値演算や論理演算を行う。

【００８６】また、該ＣＰＵは内部に各種バスや信号線を含んでいる。ＰＡ＿ＢＵＳ７２やＰＢ＿ＢＵＳ７４はＡＬＵ４０の入力信号を伝送する機能等を有する。ＷＷ＿ＢＵＳ７６はＡＬＵ４０の演算結果を取り出して汎用レジスタに伝送する機能等を有する。ＸＡ＿ＢＵＳ７８は汎用レジスタ１１やＳＰ１４から取り出したアドレスデータを伝送する機能等を有する。ＩＡ信号線８２は、ＣＰＵ内部の各部から外部のＩ＿ＡＤＤＲ＿ＢＵＳ９２へアドレスデータを伝送する。ＤＡ信号線８４は、ＣＰＵ内部の各部から外部のＤ＿ＡＤＤＲ＿ＢＵＳ９６へアドレスデータを伝送する。ＤＩＮ信号線８６は、ＣＰＵ外部のＤ＿ＤＡＴＡ＿ＢＵＳ９８からＣＰＵ内部の各部へデータを伝送する。ＤＯＵＴ信号線８８は、ＣＰＵ内部の各部から外部のＤ＿ＤＡＴＡ＿ＢＵＳ９８へデータを伝送する。ＩＡ入力切換８３はＩＡ信号線８２へ出力される各種信号（ＰＡ＿ＢＵＳ７２、ＷＷ＿ＢＵＳ７６、ＰＣ１２、ＰＣ＋２）の切換を行う。ＤＯＵＴ入力切換８９はＤＯＵＴ信号線８８へ出力される各種信号（ＰＡ＿ＢＵＳ７２、ＷＷ＿ＢＵＳ７６、ＰＣ１２、ＰＣ＋２）の切換を行う。

【００８７】また、ＣＰＵ１０の前記各部は前記命令デコーダ２０の出力する制御信号に基づき、命令の実行をおこなうもので、スタックポインタ専用命令群を、前記制御信号及び前記スタックポインタ専用レジスタの内容に基づき実行する前記実行手段としても機能する。

【００８８】本ＣＰＵ１０は、１６ビットの命令データバス（Ｉ＿ＤＡＴＡ＿ＢＵＳ）９４、命令データアクセスのための命令アドレスバス（Ｉ＿ＡＤＤＲ＿ＢＵＳ）９２と、３２ビットのデータバス（Ｄ＿ＤＡＴＡ＿ＢＵＳ）９８と、データアクセスのためのデータアドレスバス（Ｄ＿ＡＤＤＲ＿ＢＵＳ）９６と、コントロール信号のための図示しないコントロールバスを介して外部と信号のやり取りを行う。

【００８９】（２）本実施例のＣＰＵ有するレジスタセットの説明

次に本実施例のＣＰＵが有するレジスタセットの概要について、必要な部分について説明する。

【００９０】図２に本実施例のＣＰＵの持つレジスタセットを示す。本実施例のＣＰＵは、汎用レジスタ１１を１６本と、ＰＣ１２、ＰＳＲ１３、ＳＰ１４、図示しないＡＬＲ（算術演算ローレジスタ）１５、図示しないＡＨＲ（算術演算ハイレジスタ）１６を含むレジスタセットを有している。

【００９１】前記汎用レジスタ１１は機能的に等価な３２ビットのレジスタであり、Ｒ０からＲ１５と名付けられている。該汎用レジスタ１１はデータ演算時及びアドレス計算時に使用される。

【００９２】またＰＣ１２は、３２ビット長のインクリメンタルカウンタであり、現在実行中の命令のアドレスであるプログラムカウンタを保持している。本文中で、レジスタ名を指すときはＰＣといい、ＰＣに格納された値をさすときはプログラムカウンタという。

【００９３】該ＰＣ１２は、ロード命令等で直接アクセスすることは出来ない。ｃａｌｌ命令、ｉｎｔ命令や割り込み、例外が発生すると、プログラムカウンタはＰＣ１２から読み出されスタックに退避される。この様に分岐命令が実行されると飛びさきアドレスがＰＣに設定される。条件分岐命令で分岐する場合も同様である。そして、ｒｅｔ命令やｒｅｔｉ命令により、戻りさき命令アドレスがスタックより読み出され、ＰＣ１２に復帰される。

【００９４】ＰＳＲ（プロセッサステータスレジスタ）１３は、フラグが割り当てられている３２ビットレジスタであり、ＣＰＵの現在の状態を保持している。ｉｎｔ命令、割り込み、例外等が発生すると、それぞれの処理ルーチンに分岐する際に、その時のＰＳＲの状態がスタックに退避される。逆にｒｅｔｉ命令の実行で、退避されていた値がＰＳＲに復帰される。

【００９５】ＳＰ１４は３２ビットのスタックポインタ専用のレジスタで、スタックの先頭番地をさすスタックポインタが格納されている。本文中で、レジスタ名を指すときはＳＰといい、ＳＰに格納された値をさすときはスタックポインタという。但し、スタックポインタは常にワードの境界を指しているため、前記スタックポインタの下位の２ビットは常に０である。

【００９６】該スタックポインタは、本実施例で用意しているスタックポインタ専用命令群に含まれる各種命令の実行や、トラップの発生に伴い更新される。スタックポインタの更新を行うスタックポインタ専用命令としては、ｃａｌｌ命令やｒｅｔ命令等の他のルーチンへ分岐する命令、スタックポインタ移動命令、ｐｕｓｈｎ命令、ｐｏｐｎ命令等がある。例えばｃａｌｌ命令が実行されると、まずスタックポインタがワードサイズ（４）だけデクリメント（－４）され、ＰＣ１２がスタックに退避される。また、ｒｅｔ命令が実行されると、逆にスタックより戻りさきアドレスがＰＣにロードされ、スタックポインタはワードサイズ分インクリメント（＋４）される。ｉｎｔ命令が実行されたり、または、割り込み、例外等が発生したときには，スタックにＰＣやＰＳＲの値が以下の手順で退避される。

【００９７】１）ＳＰ＝ＳＰ－４

２）スタックポインタで指されるスタックの先頭番地にＰＣを退避する。

【００９８】３）ＳＰ＝ＳＰ－４

４）スタックポインタで指されるスタックの先頭番地にＰＳＲを退避する。

【００９９】ｒｅｔｉ命令が実行されると、上記とは逆の処理を行ってＣＰＵは以前の状態に戻る。このようにｃａｌｌ命令やｒｅｔ命令やｉｎｔ命令が実行されると、該実行に基づきスタックポインタが更新される。各命令の詳細については後述する。

【０１００】また、本実施例においてトラップとは、命令の実行に非同期に発生する割り込みと命令の実行により発生する例外を合わせたものをいう。トラップが発生すると、ＣＰＵはプログラムカウンタ（ＰＣ）とプロセッサステータスレジスタ（ＰＳＲ）をスタックに退避してからトラップテーブルよりベクタテーブルを読み出し、そのトラップに対応する処理ルーチンへ分岐する。トラップの発生に伴って，ＰＳＲ中のＩＥビット（割り込みイネーブル）はクリアされ、それ以上のマスク可能割り込みの発生が禁止される。再び、マスク可能な外部割り込みをイネーブルにするには，ＰＳＲに対するロード命令を使ってＰＳＲのＩＥビットに１を書き込んで設＊定する。

【０１０１】トラップ処理から元のルーチンに復帰するには、ｒｅｔｉ命令を使う。ｒｅｔｉ命令を実行すると、ＣＰＵはスタックからＰＳＲ、ＰＣの順に読み出して、ＰＳＲを元の値に戻すとともに、戻りアドレスに分岐する。なお、例外にはデバッグ例外、アドレス不整列例外、オーバーフロー例外、ゼロ除算例外等がある。

【０１０２】ＡＬＲ（算術演算ローレジスタ）１５及びＡＨＲ（算術演算ハイレジスタ）１６については説明を省略する。

【０１０３】ＣＰＵの持つ特殊レジスタＰＳＲ１３、ＳＰ１４、ＡＬＲ１５、ＡＨＲ１６はロード命令を使って汎用レジスタとの間でデータ転送を行うことが出来る。各レジスタは、特殊レジスタ番号を持っており、この番号を使ってアクセスされる。

【０１０４】

特殊レジスタ名	特殊レジスタ番号	アセンブラの記述方法
プロセッサステータスレジスタ	0	%ＰＳＲ
ＳＰ	1	%ＳＰ
算術演算ローレジスタ	2	%ＡＬＲ
算術演算ハイレジスタ	3	%ＡＨＲ

（３）スタック及びスタックポインタについての説明

スタックはメモリにもうけられた一時記憶領域で、連続した領域の下方からデータが棚状に積み重ねて記憶される。スタックポインタは、スタックメモリの一番上つまり最後に記憶されたデータのアドレスを示している。

【０１０５】スタックポインタの一般的な動作について図３を用いて説明する。

【０１０６】図３の１００はメモリに設けられたスタック領域を表している。斜線部分１０２が最後に格納されたデータであるとすると、ＳＰ１４に格納されたスタックポインタは、該データのメモリアドレス１０００を示している。なお、斜線部分１０２の下方領域１０４は既にデータが格納されている領域で、斜線部分１０２の上方領域１０６はこれからデータが格納される領域を示している。

【０１０７】スタックポインタは常にワードの境界を指しているため、スタックに情報を書き込む時はＳＰ１４の格納されたスタックポインタを４だけ上に動かして、該スタックポインタのさす場所に情報を格納する。またスタックに格納されている情報を取り出すときは、現在ＳＰ１４が示すアドレスの情報をとりだして、ＳＰ１４に格納されたスタックポインタを４だけ下に動かす。このようにスタックポインタは常に最後にスタックに格納された情報の格納アドレスを示している。

【０１０８】（４）スタックポインタ専用命令群についての説明

通常ＲＩＳＣ方式のＣＰＵでは汎用レジスタをスタックポインタとして使用するが、本実施例ではスタックポインタ専用のレジスタであるＳＰ１４を有しており、前記スタックポインタ専用命令群の操作対象となる。

【０１０９】前記スタックポインタ専用命令群は、スタックポインタ専用のレジスタであるＳＰ１４を暗黙のオペランドとし、ＳＰ１４をもちいて操作をおこなう複数の命令の総称である。該スタックポインタ専用命令群は、ＳＰ相対ロード命令（ｌｄ等）、スタックポインタ移動命令（ａｄｄ、ｓｕｂ）、サブルーチンへ分岐する命令（ｃａｌｌ等）及びリターン命令（ｒｅｔ等）等の他のルーチンへ分岐する命令、連続プッシュ命令（ｐｕｓｈｎ）、連続ポップ命令（ｐｏｐｎ）を含む。

【０１１０】これらにの命令に共通するのは、スタックポインタ専用命令であるため、オブジェクトコードにスタックポインタを特定するための情報が必要ないということである。従ってスタックポインタを用いる処理を短い命令長で効率よく記述することができるという効果も共通する。

【０１１１】また、これらの命令を用いるとメモリに設けられたスタックに記憶されている情報の処理を効率よく行うことができる。また、割り込み処理及びサブルーチンコール・リターンの処理を効率よく行うことが出来る。

【０１１２】ここで、サブルーチンがコールされる場合の前記スタックポインタ専用命令の使用例とメモリにもうけられたスタックの使用状態及びスタックポインタの状態について図４を用いて説明する。図４のメインプログラム５００とサブルーチン５２０は、Ｃコンパイラが作成したオブジェクトコードで記述されたプログラムで

ある。５４０はメモリ上のスタックの状態を示している。５０２は汎用レジスタＲ０～Ｒ３を使用する処理が行われていることを示している。５０６はサブルーチンコール命令を示している。メインプログラム５００でサブルーチンコール命令が実行されるサブルーチンコール命令前、即ち５０４に示す前の命令が実行された後スタックポインタ（ＳＰ）はメモリ上のスタック５４０上の①のアドレスを指していたとする。前記サブルーチンコール命令が実行されると、実行の制御はサブルーチン５２０にわたる。このとき本実施例では、前記スタックポインタ専用命令であるサブルーチンへ分岐する命令（ｃａｌｌ命令）が実行される。該命令が実行されると、スタックポインタ（ＳＰ）の値は自動的に－４だけインクリメントされ（図４の②ＳＰ参照）、該スタックポインタ②ＳＰが指しているスタック上のエリア５４４にメインプログラムへのリターンアドレスが格納される。

【０１１３】そして、サブルーチン５２０側では、実行の最初にメインプログラムで使用されていた汎用レジスタＲ０～Ｒ３に格納されていた値をスタックに転送する処理を行う。５２４は、汎用レジスタＲ０～Ｒ３に格納されていた値をスタックに待避する処理を行うスタックポインタ専用命令である連続プッシュ命令（ｐｕｓｈｎ）を示している。該命令が実行されると汎用レジスタＲ０～Ｒ３に格納されていた値が連続的にスタックに転送され、図４の５５０に示すように、スタック５４０に格納される。この処理の実行が終わる（５２４③）と、スタックポインタ（ＳＰ）は５４６を指している（③ＳＰ）。

【０１１４】次にサブルーチン５２０では、サブルーチンで使用するオート変数領域の確保を行う。５２６に示したａｄｄ命令は、スタックポインタ専用命令であるスタックポインタ移動命令であり、スタックポインタを上方に移動させサブルーチン５２０で使用するスタック領域を確保する。該命令が実行されるとスタックポインタ（ＳＰ）はＸバイトだけ上方の位置５４８に移動し（④ＳＰ）、注２）に示すサブルーチンが使用するオート変数の領域が確保される。

【０１１５】５２８はサブルーチン５２０でオート変数と汎用レジスタＲ０～Ｒ３を用いた処理を行うことを示している。このときスタックポインタの位置は⑤ＳＰを指しており、オート変数のロードは、スタックポインタ専用命令であるＳＰ専用ロード命令をもちいて行われる。

【０１１６】５２９は、メモリ上に格納されたオート変数Ｓ１を汎用レジスタＲ１に転送する前記ＳＰロード命令を示している。前記オート変数Ｓ１はスタックポインタ（⑤ＳＰ）からＹバイトだけオフセット値を有する場所に格納されている。前述したようにサブルーチンの５２８の処理の最中にはスタックポインタは移動しないので、オート変数のメモリアドレスは、スタックポインタ＋オフセット値で特定され、前記ＳＰ専用ロード命令を用いて効率良く汎用レジスタとのやり取りを行うことができる。

【０１１７】また、サブルーチン５２０からメインプログラム５００に戻る前には、スタックに待避されていた汎用レジスタＲ０～Ｒ３の値を汎用レジスタＲ０～Ｒ３に復帰させ、スタックポインタがメインルーチンへのリターンアドレスが格納されているエリア５４４を指すようにしなければならない。そのためにまず、５２６のスタックポインタ移動命令で移動させたスタックポインタをもとの位置に戻してやる。５３０に示したｓｕｂ命令は、スタックポインタ専用命令であるスタックポインタ移動命令であり、スタックポインタを下方に移動させる。該命令が実行されると、スタックポインタは５４６に移動する（⑥ＳＰ参照）。次にスタックに格納されている情報５５０を汎用レジスタＲ０～Ｒ３に復旧する処理を行う。５３２は、スタックの情報を汎用レジスタＲ０～Ｒ３に転送するする処理を行うスタックポインタ専用命令である連続ポップ命令（ｐｏｐｎ）を示している。該命令が実行されるとスタックに格納されていた値５５０が汎用レジスタＲ０～Ｒ３に連続的に転送され、図４の５５０に示すように、スタック５４０に格納される。この処理の実行が終わる（５３２⑦）と、スタックポインタは５４４を指している（⑦ＳＰ参照）。

【０１１８】５３４はリターン命令を示している。前記リターン命令が実行されると、実行の制御はメインプログラム５００にわたる。このとき本実施例では、前記スタックポインタ専用命令であるリターン命令（ｒｅｔ命令）が実行される。該命令が実行されると、⑦ＳＰが指すスタックエリアに格納されたメインプログラムへのリターンアドレスが示す命令に分岐する、即ちメインプログラム５００の次の命令５０７に戻る。そしてスタックポインタの値は自動的に＋４だけインクリメントされメインプログラム５００が使用するスタックの先頭エリアに移動する（図４の⑧ＳＰ参照）。

【０１１９】以下前記各スタックポインタ専用命令についてそれぞれ命令の説明及び該命令を実行するための回路構成及び実行時の動き等について詳細に説明する。

【０１２０】（５）スタックポインタ相対ロード命令

前述したようにＣコンパイラはオート変数の領域をスタックポインタに関連づけて記憶するオブジェクトコードを作成する。具体的には、Ｃコンパイラはスタックポインタからオフセットが０のところにＹバイトだけオート変数の領域を確保する仕様になっている。

【０１２１】図５は、メモリ上のオート変数をレジスタに転送する処理を説明するための図である。オート変数ａはワードデータ、オート変数ｂ、ｃはハーフワードデータ、オート変数ｄ～ｇはバイトデータである。ＳＰに格納されたスタックポインタはスタック上のオート変数が格納される領域６００の先頭領域のアドレスである１

０００を示している。オート変数ａを格納するためのエリアとして前記スタックポインタをメモリアドレスとするスタック上のエリアが、オート変数ｂ、ｃを格納するためのエリアとしてそれぞれスタックポインタ＋２、スタックポインタ＋４をメモリアドレスとするスタック上のエリアが、オート変数ｄ～ｇを格納するためのエリアとしてそれぞれスタックポインタ＋５、スタックポインタ＋６、スタックポインタ＋７、スタックポインタ＋８をメモリアドレスとするスタック上のエリアが確保されている。そして所与の処理を実行する際には、スタックポインタ＋オフセット値のメモリアドレスで特定されるオート変数と汎用レジスタ間でデータの転送処理が必要となる。

【０１２２】本実施例のＣＰＵでは、このような転送処理を短いオブジェクトコードで記述し効率良く実行するために、スタックポインタ専用命令の一つであるＳＰ相対ロード命令として、次に示す命令セットを用意している。

【０１２３】

```
ld.b　%Rd、[%sp+imm6]　…（１）
ld.ub　%Rd、[%sp+imm6]…（２）
ld.h　%Rd、[%sp+imm7]　…（３）
ld.uh　%Rd、[%sp+imm7]…（４）
ld.w　%Rd、[%sp+imm8]　…（５）
ld.b　[%sp+imm6]、%Rs　…（６）
ld.h　[%sp+imm7]、%Rs　…（７）
ld.w　[%sp+imm8]、%Rs　…（８）
```

（１）～（８）は命令コードをアセンブラで記述したものである。（１）はスタックからバイトデータをサイン拡張してレジスタに転送する命令であり、（２）はスタックからバイトデータをゼロ拡張してレジスタに転送する命令であり、（３）はスタックからハーフワードデータをサイン拡張してレジスタに転送する命令であり、（４）はスタックからハーフワードデータをゼロ拡張してレジスタに転送する命令であり、（５）はスタックからワードデータをレジスタに転送する命令であり、（６）はレジスタからバイトデータをスタックに転送する命令であり、（７）はレジスタからハーフワードデータをスタックに転送する命令であり、（８）はレジスタからワードデータをスタックに転送する命令である。

【０１２４】［％ｓｐ＋imm6］、［％ｓｐ＋imm7］、［％ｓｐ＋imm8］は、それぞれ即値オフセット情報を表しており、該即値オフセット情報に基づき実行時に生成されるオフセット値とＳＰ１４に格納されたスタックポインタの値を加算してメモリアドレスが生成される。［％ｓｐ＋imm6］はメモリ上のデータのデータサイズがバイトの時、［％ｓｐ＋imm7］はメモリ上のデータのデータサイズがハーフワードの時、［％ｓｐ＋imm8］はメモリ上のデータのデータサイズがワードの時の即値オフセット情報である。これらはいずれも、後述する図６（Ａ）に示す用にオブジェクトコード上では６ビットの即値オフセット情報６１４として記述されている。しかし命令実行時には、imm7の場合（即ちデータサイズがハーフワードの場合）、即値オフセット情報６１４は１ビット左論理シフトしてスタックポインタに加算すべきオフセット値が生成される。また、imm8の場合（即ちデータサイズがワードの場合）、即値オフセット情報６１４は２ビット左論理シフトしてスタックポインタに加算すべきオフセット値が生成される。

【０１２５】ここにおいてスタックとはメモリに設けられた一時記憶領域をいい、前記メモリアドレスで、例えば図５に示すスタック上のオート変数（ａ～ｇ）の位置を指定することができる。

【０１２６】図６（Ａ）は、前記（１）～（８）のＳＰ相対ロード命令のビットフィールド６１０である。図６（Ａ）に示すようにＳＰ相対ロード命令は、操作機能がメモリとレジスタ間のデータの転送であることを示すオペコード６１２（６ビット）、即値で指定された即値オフセット情報（６ビット）６１４と、転送対象となる汎用レジスタのレジスタ番号６１６（４ビット）を含み、１６ビットのオブジェクトコードを有している。前記オペコード６１２はＳＰ１４を操作対象とするロード命令であることを示す共通のコードと前記データサイズ、サイン拡張及びゼロ拡張の別に応じて与えられる異なるコードとを含んでいる。従ってオペコードによりＳＰ１４に格納されたスタックポインタを操作対象とすることがわかるため、オブジェクトコードのオペランドにスタックポインタに関する情報を必要としない。即値オフセット情報６１４は転送対象となるデータのスタックポインタからのオフセット値を生成するための情報である。これは、前述したようにデータサイズを問わず６ビットで指定される。転送対象となるレジスタのアドレス６１６には、前記（１）～（５）の場合はスタックからよみだされたデータが格納されるレジスタのレジスタ番号が、前記（６）～（８）の場合はスタックに書き込まれるデータが格納されているレジスタのレジスタ番号が入っている。

【０１２７】図６（Ｂ）はスタックポインタとして汎用レジスタを用いた場合に使用する汎用レジスタを操作対象とするロード命令（以下汎用ロード命令という）のオブジェクトコードのビットフィールド６２０の例を示している。

【０１２８】図６（Ｂ）に示すように汎用ロード命令は、操作機能がメモリと汎用レジスタ間のデータの転送であることを示すオペコード６２２（６ビット）、即値で指定された即値オフセット情報６２４（６ビット）とスタックポインタとして使用する汎用レジスタを特定する第一のレジスタ番号６２６（４ビット）と、転送対象となる汎用レジスタを特定する第二のレジスタ番号６２８（４ビット）を含み、２０ビットのオブジェクトコー

ドを有している。汎用マイクロコンピュータでは命令長は8ビット単位なので24ビット若しくは32ビット命令になる。

【0129】図6（A）（B）は、いずれもスタックポインタにオフセット値を加算して特定されるメモリアドレスとレジスタの間でのデータの転送処理を行う場合に使用する命令のオブジェクトコードを示しているが、同図に示すように、SP相対ロード命令は汎用ロード命令に比べて短いオブジェクトコードで記述することができる。

【0130】以下、スタックからレジスタにデータを転送する命令として（5）の命令（スタックからレジスタにワードデータを読み出す命令なので、以下ワードデータ読み出しのSP相対ロード命令という）、レジスタからスタックにデータを転送する命令として（8）の命令（レジスタからスタックにワードデータを取り込む命令なので、以下ワードデータ書き込みのSP相対ロード命令という）を例にとりこれらの命令を実行するための構成及び実行時の動作について説明する。

【0131】まず図1を用いてこれらの命令を実行するために必要なハードウエア構成について説明する。これらの命令は外部のメモリ（ROM）52よりI_DATA_BUS94を介して伝送され、CPU10の命令デコーダ20に入力される。該命令デコーダ20で、命令が解読され命令の実行に必要な図示しない各種信号が出力される。また前記即値生成器22は、データサイズに応じて前記即値オフセット情報614の左論理シフトを行い、必要に応じてサイン拡張及びゼロ拡張行い実行に使用するオフセット値を生成し、PB_BUS74に出力する。SP14はスタックポインタを格納しており、この値はアドレス加算器30の入力に接続されたXA_BUS78に出力できる。アドレス加算器30のもう一方の入力は、即値生成器22の出力であるPB_BUS74に接続されている。アドレス加算器30の出力（ADDR）は、DA信号線84を介して外部のD_ADDR_BUS96に接続されている。

【0132】バスコントロールユニット（BCU）60は、CPUから出力される各種リクエスト信号（外部バスに出力された信号等）に基づき、スタックエリアを含むメモリ（RAM、ROM）50、52とのデータの入出力を制御し、READ、WRITE制御信号を出力する。

【0133】まずワードデータ読み出しのSP相対ロード命令の実行時の動作について説明する。

【0134】ワードデータ読み出しのSP相対ロード命令が実行されると、SP14に格納されたスタックポインタの値と、前記即値オフセット情報614に基づき即値生成器22が生成したオフセット値が加算され、メモリ読み出し用のメモリアドレスが生成される。そして該メモリアドレスに基づきメモリ上の情報が読み出され、オブジェクトコードの前記レジスタ番号616で特定される汎用レジスタに転送される。

【0135】図7はワードデータ読み出しのSP相対ロード命令の動作を説明するためのフローチャート図である。

【0136】前記命令の実行の最初にSP14に格納されているスタックポインタがXA_BUS78に出力される（ステップ210）。また、即値生成器22が即値オフセット情報から生成したオフセット値immがPB_BUS74に出力される（ステップ212）。アドレス加算器30は、前記XA_BUS78上の値と前記PB_BUS74上の値を加算し、結果（ADDR）であるメモリの読み出しアドレスをDA信号線84を介してD_ADDR_BUS96に出力する（ステップ214、ステップ216）。そしてCPUからBCU60へのデータ読み出しリクエスト信号がアクティブになり、外部メモリのリードサイクルを実行する（ステップ218）。即ち前記BCU60は該リクエスト信号に基づき、前記読み出しアドレスをメモリアドレスとしてメモリからデータが読み出され、D_DATA_BUS98に出力されるよう制御する。D_DATA_BUS98上のデータはDIN信号線86を介してWW_BUS76に出力される（ステップ220）。そしてWW_BUS76上の値は命令コードの転送対象となるレジスタ（Rs／Rd）のアドレス（4ビット）616で指定されたレジスタ番号を有するレジスタ（%Rd）に格納される（ステップ222）。

【0137】次にワードデータ書き込みのSP相対ロード命令の実行時の動作について説明する。

【0138】ワードデータ書き込みのSP相対ロード命令が実行されると、SP14に格納されたスタックポインタの値と、前記即値オフセット情報614に基づき即値生成器22が生成したオフセット値が加算され、メモリ書き込み用のメモリアドレスが生成される。そしてオブジェクトコードの前記レジスタ番号616で特定される汎用レジスタに格納されている情報が、該メモリアドレスに基づき特定されるメモリ上のエリアに転送される。

【0139】図8はワードデータ書き込みのSP相対ロード命令の動作を説明するためのフローチャート図である。

【0140】前記命令の実行の最初にSP14に格納されているスタックポインタがXA_BUS78に出力される（ステップ230）。また、即値生成器22が即値オフセット情報から生成したオフセット値immがPB_BUS74に出力される（ステップ232）。アドレス加算器30は、前記XA_BUS78上の値と前記PB_BUS74上の値を加算し、結果（ADDR）であるメモリへの書き込みアドレスをDA信号線84を介してD_ADDR_BUS96に出力する（ステップ23

4、ステップ２３６）。また、命令コードの転送対象となるレジスタ（Ｒｓ／Ｒｄ）のアドレス（４ビット）６-１６で指定されたレジスタ番号を有するレジスタ（％Ｒｄ）に格納されているデータがＰＡ＿ＢＵＳ７２に出力される（ステップ２３８）。ＰＡ＿ＢＵＳ７２上のデータはＤＯＵＴ信号線８８を介して、Ｄ＿ＤＡＴＡ＿ＢＵＳ９８に出力される（ステップ２４０）。そしてＣＰＵからＢＣＵ６０へのデータ書き込みリクエスト信号がアクティブとなり、外部メモリのライトサイクルを実行する（ステップ２４２）。即ち前記ＢＣＵ６０は前記リクエスト信号に基づき、前記書き込みアドレスをメモリアドレスとして、Ｄ＿ＤＡＴＡ＿ＢＵＳ９８で転送されてきたデータをメモリ５０に書き込む動作を制御する。

【０１４１】（６）スタックポインタ移動命令

図９（Ａ）～（Ｆ）は、複数のルーチンにわたってプログラムが実行される場合の各ルーチンによるメモリ上のスタックの使用状態及びスタックポインタの状態を説明するための図である。

【０１４２】図９（Ａ）に示すＭＡＩＮルーチン２１０の所与の処理ａの実行時におけるメモリ上のスタック領域の状態及びスタックポインタ（ＳＰ１４に格納されている値）の状態を図９（Ｄ）に示している。２２２はＭＡＩＮルーチン２１０で使用するためのスタック領域を示しており、スタックポインタは２２２の先頭アドレス２３２を示している。

【０１４３】図９（Ｂ）のＳＵＢ１ルーチン２１２は、前記ＭＡＩＮルーチン２１０から呼ばれて実行されるサブルーチンである。該ＳＵＢ１ルーチン２１２の所与の処理ｂ２１３の実行時におけるメモリ上のスタック領域の状態及びスタックポインタ（ＳＰに格納されている値）の状態を図９（Ｅ）に示している。２２４はＳＵＢ１ルーチン２１２で使用するためのスタック領域を示しており、スタックポインタは２２４の先頭アドレス２３４を示している。

【０１４４】図９（Ｃ）のＳＵＢ２ルーチン２１４は、前記ＳＵＢ１ルーチン２１０から呼ばれて実行されるサブルーチンである。該ＳＵＢ２ルーチン２１４の所与の処理ｃ２１５の実行時におけるメモリ上のスタック領域の状態及びスタックポインタ（ＳＰに格納されている値）の状態を図９（Ｆ）に示している。２２６はＳＵＢ２ルーチン２１４で使用するためのスタック領域を示しており、スタックポインタは２２６の先頭アドレス２３６を示している。

【０１４５】このように複数のサブルーチンにわたって実行が行われる場合、各ルーチンで使用するスタック領域が移動するため、それに伴いスタックポインタの値を、各ルーチンで使用するスタック領域の先頭に移動することが行われる。

【０１４６】本実施例のＣＰＵでは、このようなスタックポインタの移動処理を短いオブジェクトコードで記述し効率良く実行するために、スタックポインタ専用命令の一つであるスタックポインタ移動命令として、次に示す命令セットを用意している。

【０１４７】

```
add　％ｓｐ、imm１２　　…（９）
ｓｕｂ　％ｓｐ、imm１２　　…（１０）
```

（９）（１０）は命令コードをアセンブラで記述したものである。（９）はＳＰ１４に格納されたスタックポインタに対する即値加算命令であり、（１０）はＳＰ１４に格納されたスタックポインタに対する即値減算命令である。imm１２は、命令のオブジェクトコードに含まれた１０ビットの即値を２ビット左方向にシフトした後、ゼロ拡張されて３２ビットデータとなり、ＳＰ１４に格納されたスタックポインタとの演算に用いられる。

【０１４８】図１０（Ａ）は、前記（９）（１０）のスタックポインタ移動命令のビットフィールド６３０である。図１０（Ａ）に示すようにスタックポインタ移動命令は、操作機能がＳＰ１４に格納されたスタックポインタへの移動情報の加算及び減算であることを示すオペコード６３２（６ビット）、即値で指定された即値移動情報６３４（１０ビット）を含み、１６ビットのオブジェクトコードを有している。前記オペコード６３２はＳＰ１４を操作対象とするスタックポインタ移動命令であることを示す共通のコードと加算及び減算の別に応じて与えられる異なるコードとを含んでいる。従ってオペコードによりＳＰ１４に格納されたスタックポインタを操作対象とすることがわかるため、オブジェクトコードのオペランドにスタックポインタに関する情報を必要としない。即値移動情報６３４はスタックポインタに加算又は減算を行うオフセット値を生成するための情報である。

【０１４９】図１０（Ｂ）はスタックポインタとして汎用レジスタを用いた場合に使用する加算及び減算命令（以下汎用演算命令という）のオブジェクトコードのビットフィールド６４０の例を示している。

【０１５０】図１０（Ｂ）に示すように汎用演算命令は、操作機能が汎用レジスタの値への即値演算情報の加算及び減算であることを示すオペコード６４２（６ビット）、即値で指定された即値演算情報６４４（１０ビット）と操作対象となる汎用レジスタを特定するレジスタ番号６４６（４ビット）とを含み、２０ビットのオブジェクトコードを有している。汎用マイクロコンピュータでは命令長は８ビット単位なので２４ビット若しくは３２ビット命令になる。

【０１５１】図１０（Ａ）（Ｂ）は、いずれもスタックポインタに即値を加算及び減算する処理を行う場合に使用する命令のオブジェクトコードであるが、同図に示すように、スタックポインタ移動命令は汎用演算命令に比べて短いオブジェクトコードで記述することができる。

【０１５２】以下、（９）のＳＰに対する即値加算命令（以下加算スタックポインタ移動命令という）と、（１

０）のＳＰに対する即値減算命令（以下減算スタックポインタ移動命令という）を実行するための構成及び実行時の動作について説明する。

【０１５３】まず図１を用いてこれらの命令を実行するために必要なハードウエア構成について説明する。これらの命令は外部のメモリ（ＲＯＭ）５２よりＩ＿ＤＡＴＡ＿ＢＵＳ９４を介して伝送され、ＣＰＵ１０の命令デコーダ２０に入力される。該命令デコーダ２０で、命令が解読され命令の実行に必要な図示しない各種信号が出力される。また前記即値生成器２２は、前記即値移動情報６３４の１０ビットを2ビット左論理シフトし、ゼロ拡張してＰＡ＿ＢＵＳ７２に出力する。ＳＰ１４はスタックポインタを格納しており、この値はＸＡ＿ＢＵＳ７８に出力される。ＸＡ＿ＢＵＳ７８はＡＬＵ４０の入力となるＰＢ＿ＢＵＳ７４に接続されている。ＡＬＵ４０のもう一方の入力は、即値生成器２２の出力であるＰＡ＿ＢＵＳ７２に接続されている。ＡＬＵ４０の出力は、ＷＷ＿ＢＵＳ７６に接続されている。ＷＷ＿ＢＵＳ７６はＳＰの入力に接続されている。

【０１５４】まず加算スタックポインタ移動命令の実行時の動作について説明する。

【０１５５】加算スタックポインタ移動命令が実行されると、ＳＰ１４に格納されたスタックポインタの値と、前記即値移動情報６３４に基づき即値生成器２２が生成した移動即値が加算されて新たなスタックポインタが生成され、その値がＳＰ１４に格納される。

【０１５６】図１１は加算スタックポインタ移動命令の動作を説明するためのフローチャート図である。

【０１５７】前記命令の実行の最初にＳＰ１４に格納されているスタックポインタがＸＡ＿ＢＵＳ７８に出力される（ステップ２５０）。そして前記ＸＡ＿ＢＵＳ７８上のデータはＰＢ＿ＢＵＳ７４に出力される（ステップ２５２）。また、即値生成器２２が即値移動情報に基づき生成した移動即値ｉｍｍがＰＡ＿ＢＵＳ７２に出力される（ステップ２５４）。ＡＬＵ４０は、前記ＰＢ＿ＢＵＳ７４上の値と前記ＰＡ＿ＢＵＳ７２上の値を加算し、結果をＷＷ＿ＢＵＳ７６に出力する（ステップ２５６）。そして、ＷＷ＿ＢＵＳ７６上の値がＳＰ１４に入力される（ステップ２５８）。

【０１５８】次に減算スタックポインタ移動命令の実行時の動作について説明する。

【０１５９】減算スタックポインタ移動命令が実行されると、ＳＰ１４に格納されたスタックポインタの値から、前記即値移動情報６３４に基づき即値生成器２２が生成した移動即値が減算されて新たなスタックポインタが生成され、その値がＳＰ１４に格納される。

【０１６０】図１２は減算スタックポインタ移動命令の動作を説明するためのフローチャート図である。

【０１６１】前記命令の実行の最初にＳＰ１４に格納されているスタックポインタがＸＡ＿ＢＵＳ７８に出力される（ステップ２６０）。そして前記ＸＡ＿ＢＵＳ７８上のデータはＰＢ＿ＢＵＳ７４に出力される（ステップ２６２）。また、即値生成器２２が即値移動情報に基づき生成した移動即値ｉｍｍがＰＡ＿ＢＵＳ７２に出力される（ステップ２６４）。ＡＬＵ４０は、前記ＰＢ＿ＢＵＳ７４上の値から前記ＰＡ＿ＢＵＳ７２上の値を減算し、結果をＷＷ＿ＢＵＳ７６に出力する（ステップ２６６）。そして、ＷＷ＿ＢＵＳ７６上の値がＳＰ１４に入力される（ステップ２６８）。

【０１６２】（７）分岐命令

図１３は、ｃａｌｌ命令とｒｅｔ命令によるプログラムの実行の制御を説明するための図である。図１３に示すように、ＭＡＩＮルーチン３００において、サブルーチンＳＵＢ３１０へ分岐するためのｃａｌｌ命令が実行されると（３０２）、制御がサブルーチンＳＵＢ３１０に渡る。サブルーチンの最後には、ｒｅｔ命令（３１２）が書かれており、この命令が実行されると、ＭＡＩＮルーチン３００の前記ｃａｌｌ命令（３０２）の次の命令（３０４）に戻る。従って、図１３に示す場合❶❷❸の順でプログラムが実行されることになる。このようにサブルーチンＳＵＢ３１０での実行が終わるとＭＡＩＮルーチン３００に戻り、前記ｃａｌｌ命令（３０２）の次の命令（３０４）から実行するため、サブルーチンＳＵＢ３１０に分岐する際、どこかに戻り先のアドレスを記憶しておくことが必要となる。このため、ｃａｌｌ命令等のサブルーチンへ分岐する分岐命令実行の際には、図４で説明したように戻り先アドレスをスタックに待避する処理が行われ、ｒｅｔ命令等のサブルーチンから戻る分岐命令の実行の際には前記スタックから戻り先アドレスをプログラムカウンタに戻す処理（以下プログラムカウンタの待避及び復旧の処理という）が行われる。

【０１６３】従来のＲＩＳＣ方式のＣＰＵでは、前記プログラムカウンタの待避及び復旧の処理をソフトウエア的に実現していたため、ｃａｌｌ命令等の分岐命令を実行する際には、これらの処理を実行するためオブジェクトコード（アセンブラ命令）も必要であった。例えばｃａｌｌ命令を実行する際には、スタックポインタをワードサイズ（４）だけデクリメント（－４）して、プログラムカウンタの値に基づきｃａｌｌ命令の次の命令のアドレスをスタックに格納するためのオブジェクトコード（アセンブラ命令）が必要であった。

【０１６４】しかし本実施例のＣＰＵは、前記ｃａｌｌ命令やｒｅｔ命令が実行されると、前記プログラムカウンタの待避及び復旧の処理も一緒に行うようなハードウエア構成を有している。従って、前記ｃａｌｌ命令やｒｅｔ命令と別に、前記プログラムカウンタの待避及び復旧の処理を記述するオブジェクトコード（アセンブラ命令）を必要としない。本実施例のＣＰＵでは、このような前記プログラムカウンタの待避及び復旧の処理を一命令で実行するために、スタックポインタ専用命令の一つ

である分岐命令として次に示す命令セットを用意している。

【０１６５】
```
call  sign9        …（１１）
call  %Rb          …（１２）
ret                …（１３）
reti               …（１４）
retd               …（１５）
int   imm2         …（１６）
brk                …（１７）
```

（１１）～（１７）は命令コードをアセンブラで記述したものである。（１１）はＰＣ相対サブルーチンコール命令で、プログラムカウンタＰＣをベースアドレスとして、オペランドで指定されたディスプレースメント情報sign9に基づき分岐先アドレスを相対的に指定して分岐するコール命令である。（１２）はレジスタ間接サブルーチンコール命令であり、オペランドで指定されたレジスタに格納されている分岐先アドレスに分岐するコール命令である。（１３）はサブルーチンからのリターン命令である。（１４）は割り込み又は例外処理ルーチンからのリターン命令である。（１５）はデバッグ処理ルーチンからのリターン命令である。（１６）はソフトウエア割り込み命令である。（１７）はソフトウエア・デバッグ割り込み命令である。

【０１６６】図１４は、前記（１１）のＰＣ相対サブルーチンコール命令のビットフィールド６５０である。図１４に示すようにＰＣ相対サブルーチンコール命令は、操作機能がプログラムカウンタをベースアドレスとして、分岐先アドレスを相対的に指定してサブルーチンへ分岐するコール命令であることを示すオペコード６５２（８ビット）、即値で指定されたディスプレースメント情報sign9（８ビット）６５４とを含み、１６ビットのオブジェクトコードを有している。前記８ビットの即値は、実行時に１ビット左に論理シフトされた後、サイン拡張される。

【０１６７】本実施例のＣＰＵでは、ｃａｌｌ命令実行には図１４に示すオブジェクトコードのみで、プログラムカウンタのスタックへの待避も実行することが出来る。

【０１６８】以下、サブルーチンへ分岐する命令として（１１）のＰＣ相対サブルーチンコール命令、サブルーチンからリターンする命令として（１３）のリターン命令を例にとりこれらの命令を実行するための構成及び実行時の動作について説明する。

【０１６９】まず図１を用いてこれらの命令を実行するために必要なハードウエア構成について説明する。これらの命令は外部のメモリ（ＲＯＭ）５２よりＩ＿ＤＡＴＡ＿ＢＵＳ９４を介して伝送され、ＣＰＵ１０の命令デコーダ２０に入力される。該命令デコーダ２０で、命令が解読され命令の実行に必要な図示しない各種信号が出力される。また前記即値生成器２２は、前記ディスプレースメント情報６５４を１ビット左に論理シフトした後サイン拡張して３２ビットの即値ディスプレースメントｉｍｍを生成し、ＰＢ＿ＢＵＳ７４に出力する。ＳＰ１４はスタックポインタを格納しており、この値はアドレス加算器３０の入力に接続されたＸＡ＿ＢＵＳ７８に出力できる。アドレス加算器３０のもう一方の入力は、即値生成器２２の出力であるＰＢ＿ＢＵＳ７４に接続されている。アドレス加算器３０の出力（ＡＤＤＲ）は、ＩＡ信号線８２を介して外部のＩ＿ＡＤＤＲ＿ＢＵＳ９２に接続されている。

【０１７０】また、Ｉ＿ＡＤＤＲ＿ＢＵＳ９２及びＩ＿ＤＡＴＡ＿ＢＵＳ９４は命令のオブジェクトコードが格納されたＲＯＭ５２に接続されており、バスコントロールユニット（ＢＣＵ）６０は、ＣＰＵから出力される各種リクエスト信号（外部バスに出力された信号等）に基づき、メモリ（ＲＯＭ）５２から前記命令のオブジェクトコードを読み出すＲＥＡＤ制御信号を出力する。

【０１７１】まずＰＣ相対サブルーチンコール命令の実行時の動作について説明する。

【０１７２】ＰＣ相対サブルーチンコール命令が実行されると、図４で説明したように、ＰＣ１２に格納されたプログラムカウンタの値がスタックに待避され、ＳＰ１４に格納されたスタックポインタの値がワードサイズ（４）だけディクリメントされる。そしてＰＣ１２にプログラムカウンタと前記３２ビットの即値ディスプレースメントを加算して得られた分岐先アドレスがセットされる。

【０１７３】図１５はＰＣ相対サブルーチンコール命令の動作を説明するためのフローチャート図である。

【０１７４】前記命令の実行の最初にＳＰ１４に格納されているスタックポインタがＸＡ＿ＢＵＳ７８に出力される（ステップ２７０）。これはアドレス加算器の一方の入力となり、即値生成器２２が生成したconstantデータ（－４）がアドレス加算器３０の他方の入力となる。そしてアドレス加算器３０で前記ＸＡ＿ＢＵＳ７８上のスタックポインタの値と－４を加算して戻りアドレスを格納するスタックへの書き込みアドレス生成され、ＷＷ＿ＢＵＳ７６に出力される。また、前記書き込みアドレスはＤＡ信号線８４を介してＤ＿ＡＤＤＲ＿ＢＵＳ９６に出力される。（ステップ２７２）。また、ＰＣインクリメンタ４４ではＰＣ１２に格納されたプログラムカウンタの値が＋２されて戻りアドレスが生成されて、ＤＯＵＴ信号線８８を介してＤ＿ＤＡＴＡ＿ＢＵＳ９８に出力される（ステップ２７４）。そしてＣＰＵからＢＣＵ６０へのデータ書き出しリクエスト信号がアクティブになり、外部メモリのライトサイクルを実行する（ステップ２７６）。即ち前記ＢＣＵ６０は該リクエスト信号に基づき、前記書き込みアドレスをメモリアドレスとしてメモリに設けられたスタックに前記戻りアドレスが格納される。

【0175】そして、WW_BUS76上の値がSP14に出力される（ステップ278）。すなわちスタックポインタの値が－4した値に更新される。

【0176】次にPC12に格納されているプログラムカウンタがXA_BUS78に出力される（ステップ280）。また、前記即値生成器22は、命令のオブジェクトコードに含まれている前記ディスプレースメント情報654を1ビット左に論理シフトした後サイン拡張して32ビットの即値ディスプレースメントimmを生成し、PB_BUS74に出力する。前記アドレス加算器30はXA_BUS78上のプログラムカウンタとPB_BUS74上の即値ディスプレースメントimmを加算し分岐アドレス（ADDR）を生成し、IA信号線82を介して、I_ADDR_BUS92に出力する（ステップ282）。そしてCPUからBCU60への命令読み出しリクエスト信号がアクティブとなり、外部メモリ（ROM）52のリードサイクルを実行し、分岐先の命令のオブジェクトコードを読み出す（ステップ284）。

【0177】次にret命令の実行時の動作について説明する。

【0178】ret命令が実行されると、図4で説明したように、スタックに待避されていたプログラムカウンタの値がPC12に復帰され、SP14に格納されたスタックポインタの値がワードサイズ（4）だけインクリメントされる。

【0179】図16はret命令の動作を説明するためのフローチャート図である。

【0180】前記命令の実行前には、SP14に格納されたスタックポインタは呼ばれたルーチンへの戻り先アドレスが格納されたスタックのアドレスをさしている。

【0181】前記命令の実行の最初にSP14に格納されているスタックポインタがXA_BUS78に出力される（ステップ290）。そしてXA_BUS78上のスタックポインタはD_ADDR_BUS96に出力される（ステップ292）。そしてCPUからBCU60へのデータ読み込みリクエスト信号がアクティブになり、外部メモリのリードサイクルを実行する。即ち前記BCU60は該リクエスト信号に基づき、前記スタックポインタをメモリアドレスとしてメモリに設けられたスタックから前記戻りアドレスを読み出す。メモリ（RAM）50から読み出された前記戻り先アドレスはD_DATA_BUS94からDIN信号線86を介してCPU内部にとりこまれ、さらにDIN信号線からIA信号線82を介してI_ADDR_BUS92に出力される（ステップ294）。そしてCPUからBCU60への命令読み出しリクエスト信号がアクティブとなり、外部メモリ（ROM）52のリードサイクルを実行し、戻り先アドレスの命令のオブジェクトコードを読み出す（ステップ296）。

【0182】そして、XA_BUS78上のスタックポインタの値は、前記アドレス加算器30の一方の入力となる。また即値生成器22が生成したconstantデータ（＋4）がアドレス加算器30の他方の入力となる。そしてアドレス加算器30で前記XA_BUS78上のスタックポインタの値と＋4を加算して、戻り先のルーチンが確保したスタック領域の先頭エリアのアドレスが生成され、WW_BUS76に出力される（ステップ298）。そして、WW_BUS76上の前記アドレス（戻り先のルーチンが確保したスタック領域の先頭エリアのアドレス）がSP14に出力される（ステップ300）。

【0183】（8）連続プッシュ命令（pushn）、連続ポップ命令（popn）の説明

前述したように、最近の特にRISC方式のCPUは、性能を高めるため内部に多くの汎用レジスタを持ち、メモリにアクセスすることなくCPU内部で高速に多くの処理をおこなうよう構成されている。本実施例でも内部に16本の汎用レジスタをもち、処理の高速化を図っている。しかし、このように内部レジスタを多く持つと，割り込み処理やサブルーチンコールの時のレジスタ退避と復旧の処理の際、退避すべきレジスタ数が多くなる。

【0184】このようなレジスタの待避や復旧を行う場合、従来は、番地部で指定した内容をスタックに格納するpush命令や、スタックの内容をレジスタに取り出すpop命令を用いていた。ここで一般的なpush命令及びpop命令時の動きを説明する。

【0185】図17（A）（B）はpush命令実行時の動きを模式的に示した図であり、図18（A）（B）はpop命令実行時の動きを模式的に示した図である。図17（A）（B）及び図18（A）（B）を用いて、複数の汎用レジスタとスタック間でデータの転送を行う場合の動きを説明する。

【0186】図17（A）は、汎用レジスタR1の内容をスタックに書き出す命令である’push R1’が実行された場合の動きを示している。該命令の実行時には、SP14の内容は現在の値から4がひかれた値に更新される（図17（A）に示すように1000が996に更新される）。そして、更新されたSP14のスタックポインタが示すメモリアドレスである996に汎用レジスタR1の内容aが書き込まれる。

【0187】図17（B）は、さらに汎用レジスタR2の内容をスタックに書き出す命令である’push R2’が実行された場合の動きを示している。該命令の実行時には、SP14の内容は現在の値から4がひかれた値に更新される（図17（B）に示すように996が992に更新される）。そして、更新されたSP14のスタックポインタが示すメモリアドレスである992に汎用レジスタR2の内容bが書き込まれる。

【0188】図18（A）は、スタックの内容を汎用レ

ジスタＲ２に取り出す命令である’ｐｏｐ　Ｒ2’が実行された場合の動きを示している。該命令の実行時には、ＳＰ１４のスタックポインタが示しているメモリアドレス９９２に格納されている内容ｂがとりだされて汎用レジスタＲ２に格納される。そしてＳＰＩ４の内容は現在の値に４が足された値に更新される（図１８（Ａ）示すように実行前９９２であったのが実行後９９６に更新される）。

【０１８９】図１８（Ｂ）は、更にスタックの内容を汎用レジスタＲ１に取り出す命令である’ｐｏｐ　Ｒ１’が実行された場合の動きを示している。該命令の実行時には、ＳＰ１４のスタックポインタが示しているメモリアドレス９９６に格納されている内容ａがとりだされて汎用レジスタＲ１に格納される。そしてＳＰ１４の内容は現在の値に４が足された値に更新される（図１８（Ｂ）に示すように実行前９９６であったのが実行後１０００に更新される）。

【０１９０】このように従来は、複数の汎用レジスタとスタックでデータの転送を行う場合、ｐｕｓｈ命令やｐｏｐ命令を複数回繰り返して実行することが必要であった。ｐｕｓｈ命令やｐｏｐ命令は１回の命令で１本のレジスタしか操作することができなかったからである。

【０１９１】従って多数のレジスタに対してスタックへの待避やスタックからの復旧処理を行う場合、命令数の増加によりオブジェクトコードのサイズの増大を招いていた。また、プログラムの実行ステップも多くなり、プログラムの実行時間や処理動作の遅延を招いていた。

【０１９２】そこで 本実施例のＣＰＵでは、次に示す命令セットを用意している。

【０１９３】ｐｕｓｈｎ　　%Ｒｓ　　…（１８）
ｐｏｐｎ　　%Ｒｄ　　…（１９）

（１８）は連続プッシュ命令のアセンブラの記述を示しており、%ＲｓからＲ０までのｎ個（ｎは１から１６の自然数）の汎用レジスタの内容を連続的にスタックへプッシュする命令である。（１９）は連続ポップ命令のアセンブラの記述を示しており、スタックからｎ個（ｎは１から１６の自然数）のワードデータを連続的に%ＲｄからＲ０までの汎用レジスタへポップする命令である。ｐｕｓｈｎ、ｐｏｐｎ命令は、いずれもオペコードとオペランドからなる。%Ｒｓはｐｕｓｈｎ命令のオペランドでレジスタ%ＲｓからＲ０までをスタックに書き込む場合のＲｓのレジスタ番号を示している。%Ｒｄはｐｏｐｎ命令のオペランドでＲ０から%Ｒｄ迄のレジスタにスタックからデータを持ってくる場合のＲｄのレジスタ番号を示している。

【０１９４】図１９にｐｕｓｈｎ、ｐｏｐｎ命令のビットマップを示す。下位の４ビットのフィールドに%Ｒｓまたは%Ｒｄを示すコードが入り、１６本ある汎用レジスタの任意のレジスタが指定できる。特殊レジスタとスタックの間でデータの転送をするときは、汎用レジスタを介して行う。

【０１９５】図２０は連続プッシュ命令（ｐｕｓｈｎ）、連続ポップ命令（ｐｏｐｎ）を実行するためのハードウェア構成を説明するためのブロック図である。図１から連続プッシュ命令（ｐｕｓｈｎ）及び連続ポップ命令（ｐｏｐｎ）の説明に必要な部分を取り出して、説明に必要な部分を追加した構成になっている。図１と同一部分を指すものについて同一の番号を付している。図中１１はＲ０からＲ１５の１６本の汎用レジスタである。汎用レジスタ１１はデータバス（Ｄ＿ＤＡＴＡ＿ＢＵＳ）９８とデータの入出力を行う。またレジスタを選択するレジスタ選択アドレス信号５４はコントロール回路ブロック４５から来る。１４はＳＰでスタックポインタが格納されている。ＳＰ１４の値はアドレスバス（Ｄ＿ＡＤＤＲ＿ＢＵＳ）９６と３２ビットのアドレス加算器３０の入力に接続する内部アドレスバス（ＸＡ＿ＢＵＳ）７８に出力できる。アドレス加算器３０のもう一方の入力はコントロール回路ブロック４５からの出力されるオフセット信号２４に接続されている。アドレス加算器３０の出力はラッチ（Ａｄｄ＿ＬＴ）３２でラッチされ、さらにＸＡ＿ＢＵＳ７８またはＷＷ＿ＢＵＳ７６に出力される。ＷＷ＿ＢＵＳ７６はＳＰ１４の入力に接続されている。コントロール回路ブロック４５の中には４ビットのカウンタ（ｃｏｕｎｔｘ）４６があり、転送するレジスタ数をカウントする。又、コントロール回路ブロック４５中には図２０では示されていないがインストラクションレジスタがあり、ｐｕｓｈｎ、ｐｏｐｎ命令のオペランド%Ｒｓ、%Ｒｄを保持するとともに、インストラクションデコーダによって各命令に応じた制御信号を出力する。図中６０はバスコントロールユニット（ＢＣＵ）で外部のスタックエリアを含むメモリ（ＲＡＭ）５０とのデータの入出力を制御し、ＲＥＡＤ、ＷＲＩＴＥ制御信号を出力する。

【０１９６】まず連続プッシュ命令（ｐｕｓｈｎ）の実行時の動作について説明する。

【０１９７】連続プッシュ命令（ｐｕｓｈｎ）が実行されると、図４で説明したように、%Ｒｓのレジスタ番号の汎用レジスタから汎用レジスタＲ０までの汎用レジスタに格納された内容が連続的にスタックにプッシュされる。

【０１９８】図２１はｐｕｓｈｎ命令の動作を説明するためのフローチャート図である。

【０１９９】ｐｕｓｈｎ命令の実行の最初にオフセット信号２４のoffsetは－４になる。カウンタ（ｃｏｕｎｔｘ）４６はゼロにクリアされ、ＳＰ１４に格納されたスタックポインタ値がＸＡ＿ＢＵＳ７８に出力される（ステップ１００）。

【０２００】次にアドレス加算器３０はＸＡ＿ＢＵＳ７８上の値と－４を加算し結果をラッチ（Ａｄｄ＿ＬＴ）３２に入れる（ステップ１０１）。

【0201】ラッチ（Add_LT）32の値はアドレスバスD_ADDR_BUS96に出力される。コントロール回路ブロック45ではインストラクションレジスタの下位4ビットに保持された%Rsとカウンタ（countx）46の差を計算し結果をレジスタ選択アドレス信号54に出力する。54によって選択されたレジスタをデータバス（D_DATA_BUS）98にのせ、外部メモリ（RAM）50の書き込み動作をする（ステップ102）。

【0202】ステップ103では、カウンタ（countx）46と%Rsを比較する。等しいときはレジスタの外部メモリへの書き込みは完了しており、ラッチ（Add_LT）32の値をWW_BUS76を通じてSP14に書き込んでpushnの実行を終わる（ステップ104）。

【0203】等しくないときはカウンタ（countx）をプラス1し、ラッチ（Add_LT）32の値をXA_BUS78に出力し、ステップ101以降の処理を繰り返す（ステップ105）。

【0204】次に連続ポップ命令（popn）の実行時の動作について説明する。

【0205】連続ポップ命令（popn）が実行されると、図4で説明したように、スタックの内容が汎用レジスタR0から%Rdのレジスタ番号の汎用レジスタに連続的にブッシュされる。

【0206】図22は、popn命令の動作を説明するためのフローチャート図である。

【0207】popn命令の最初にオフセット信号24は+4になる。カウンタ（countx）46はゼロクリアされ、SP14のスタックポインタの値がXA_BUS78とアドレスバス（D_ADDR_BUS）96に出力される（ステップ110）。

【0208】次にアドレス加算器30はXA_BUS78上の値と+4を加算し結果をラッチ（Add_LT）32に入れる（ステップ111）。

【0209】次に外部メモリのリードサイクルを実行する。リードされたデータはデータバス（D_DATA_BUS）98を通じて汎用レジスタ11に書き込まれる。このときコントロール回路ブロック45ではカウンタ（countx）46をレジスタ選択アドレス信号54に出力する（ステップ112）。

【0210】ステップ113では、カウンタ（countx）46と%Rsを比較する。等しいときはレジスタの外部メモリへの書き込みは完了しており、ラッチ（Add_LT）32の値をWW_BUS76を通じてSP14に書き込んでpopnの実行を終わる（ステップ114）。

【0211】等しくないときはカウンタ（countx）をプラス1し、ラッチ（Add_LT）32の値をXA_BUS78とアドレスバス（D_ADDR_BUS）96に出力し、ステップ111以降の処理を繰り返す（ステップ115）。

【0212】このように' pushn %Rs' で%RsからR0までのレジスタをスタックに書き込め，また' popn %Rd' でスタックから必要な個数のデータをR0から%Rdまでのレジスタに戻すことが出来る。たとえば、' pushn%R3' の実行によりレジスタR3からR0までを一命令でブッシュできる。

【0213】ここにおいてレジスタの使い方に一つの制約を加えることで、更に有効な効果が発生する。即ち、レジスタの待避や復旧は割り込み処理やサブルーチンコール時等のプログラムが他のルーチンに分岐するときに特に必要になるが、このとき他の割り込みルーチンまたはサブルーチン等の呼び出されるルーチンではレジスタをR0から順に使って行くことが好ましい。このようにすると、R0からRdまたはRsに退避の必要のないレジスタが含まれず、pushn命令及びpopn命令を用いて、効率的にレジスタの待避又は復帰を行うことができる。本実施例ではレジスタはどれも同じ機能を持っており、レジスタの使い方と命令には何の制約も無いので、このような制約は何の問題もなく満たすことが出来る。

【0214】従って本実施例のpushn命令及びpopn命令を用いることにより、レジスタからメモリ中のスタックへの退避、またスタックからレジスタへのデータの復旧がpushn、popnの各々一命令で実行できる。このためオブジェクトコードサイズやプログラム実行ステップを最少にし、かつ実行サイクルを一回の命令フェッチと必要な回数のデータの転送だけですませ、最少のサイクルで行うことが出来る。これにより割り込み処理ルーチンやサブルーチンの処理も高速化を図ることができる。

【0215】一方、これを実現する構成要素はカウント手段と簡単なシーケンス制御のみであり、少ないゲート数で実現出来、ワンチップのマイクロコンピュータに適したものである。

【0216】また、本実施例ではスタックポインタ専用のレジスタSP14を用いた場合の連続ブッシュ命令（pushn）、連続ポップ命令（popn）を実行させるための構成について説明したが、スタックポインタとして汎用レジスタを使用する場合にも適用可能である。

【0217】（実施例2）図23は、本実施例のマイクロコンピュータのハードウエアブロック図である。

【0218】該マイクロコンピュータ2は、32ビットマイクロコントローラであり、CPU10とROM52とRAM50、高周波発振回路910、低周波発振回路920、リセット回路930、ブリスケーラ940、16ビットプログラマブルタイマ950、8ビットプログラマブルタイマ960、クロックタイマ970、インテ

リジェントＤＭＡ９８０、高速ＤＭＡ９９０、割り込みコントローラ８００、シリアルインターフェース８１０、バスコントロールユニット（ＢＣＵ）６０、Ａ／Ｄ変換器８３０、Ｄ／Ａ変換器８４０、入力ポート８５０、出力ポート８６０、Ｉ／Ｏポート８７０、及びそれらを接続する各種バス９２、９４、９６、９８、各種ピン８９０等を含む。

【０２１９】前記ＣＰＵ１０は、スタックポインタ専用レジスタであるＳＰを有し、前述した各種のスタックポインタ専用命令の解読、実行を行う。該ＣＰＵ１０は、前述した、実施例１の構成を有しており、前記解読手段、前記実行手段として機能する。

【０２２０】従って本実施例のマイクロコンピュータは、スタックポインタを取り扱う処理を、短い命令長で効率よく記憶し、実行することができる。

【０２２１】また、レジスタ退避やレジスタ復旧の処理を効率よく記憶し、割り込み処理及びサブルーチンコール・リターンの処理を高速に行うことができる。

【０２２２】本発明のマイクロコンピュータは例えばプリンター等のパソコン周辺機器や、携帯機器等の各種の電子機器に適用可能である。この様にすると、簡単な構成でメモリの使用効率がよく高速に処理の行える情報処理回路を内蔵することができるため、安価で高機能な電子機器を提供することが出来る。

【０２２３】なお本発明は、上記実施例で説明したものに限らず、種々の変形実施が可能である。

【０２２４】

【図面の簡単な説明】

【図１】本実施例のＣＰＵの回路構成の概略を説明するための図である。

【図２】本実施例のＣＰＵの持つレジスタセットを示す。

【図３】スタックポインタの一般的な動作について説明するための図である。

【図４】スタックポインタ専用命令の使用例とメモリにもうけられたスタックの使用状態及びスタックポインタの状態について説明するための図である。

【図５】メモリ上のオート変数をレジスタに転送する処理を説明するための図である。

【図６】図６（Ａ）（Ｂ）は、ＳＰ相対ロード命令及び汎用ロード命令のビットフィールドを示した図である。

【図７】ワードデータ読み出しのＳＰ相対ロード命令の動作を説明するためのフローチャート図である。

【図８】ワードデータ書き込みのＳＰ相対ロード命令の動作を説明するためのフローチャート図である。

【図９】図９（Ａ）～（Ｆ）は、複数のルーチンにわたってプログラムが実行される場合の各ルーチンによるメモリ上のスタックの使用状態及びスタックポインタの状態を説明するための図である。

【図１０】図１０（Ａ）（Ｂ）は、スタックポインタ移動命令及び汎用演算命令のビットフィールドを示した図である。

【図１１】加算スタックポインタ移動命令の動作を説明するためのフローチャート図である。

【図１２】減算スタックポインタ移動命令の動作を説明するためのフローチャート図である。

【図１３】ｃａｌｌ命令とｒｅｔ命令によるプログラムの実行の制御を説明するための図である。

【図１４】ＰＣ相対サブルーチンコール命令のビットフィールドを示した図である。

【図１５】ＰＣ相対サブルーチンコール命令の動作を説明するためのフローチャート図である。

【図１６】ｒｅｔ命令の動作を説明するためのフローチャート図である。

【図１７】図１７（Ａ）（Ｂ）はｐｕｓｈ命令実行時の動きを模式的に示した図である。

【図１８】図１８（Ａ）（Ｂ）はｐｏｐ命令実行時の動きを模式的に示した図である。

【図１９】ｐｕｓｈｎ、ｐｏｐｎ命令のビットマップを示す。

【図２０】連続プッシュ命令（ｐｕｓｈｎ）、連続ポップ命令（ｐｏｐｎ）を実行するためのハードウェア構成を説明するためのブロック図である。

【図２１】ｐｕｓｈｎ命令の動作を説明するためのフローチャート図である。

【図２２】ｐｏｐｎ命令の動作を説明するためのフローチャート図である。

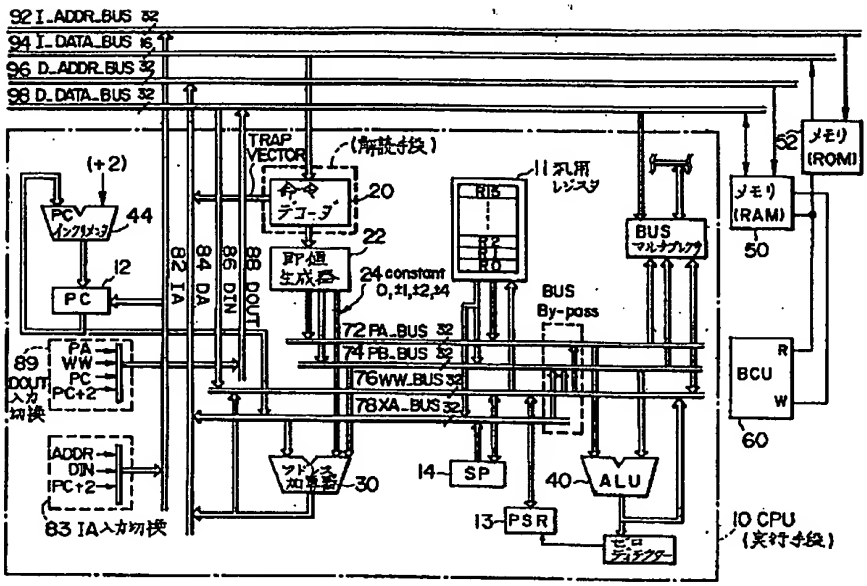
【図２３】本実施の形態のマイクロコンピュータのハードウエアブロック図である。

【符号の説明】

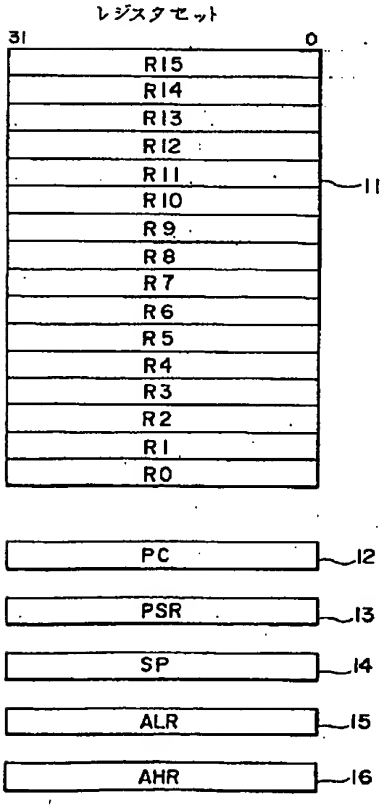
２　マイクロコンピュータ
１０　ＣＰＵ
１１　汎用レジスタ
１２　ＰＣ（プログラムカウンタ）
１３　ＰＳＲ（プロセッサステータスレジスタ）
２０　命令デコーダ
２２　即値生成器期
２４　オフセット信号
３０　アドレス加算器
３２　ラッチ（Ａｄｄ＿ＬＴ）
４０　ＡＬＵ
４４　ＰＣインクリメンタ
４５　コントロール回路ブロック
４６　４ビットカウンタ（ｃｏｕｎｔｘ）
５０　メモリ（ＲＡＭ）
５２　メモリ（ＲＯＭ）
５４　レジスタ選択アドレス信号
６０　バスコントロールユニット（ＢＣＵ）
７２、７４、７６、７８　内部バス
８２、８４、８６、８８　内部信号線

92、94、96、98　外部バス

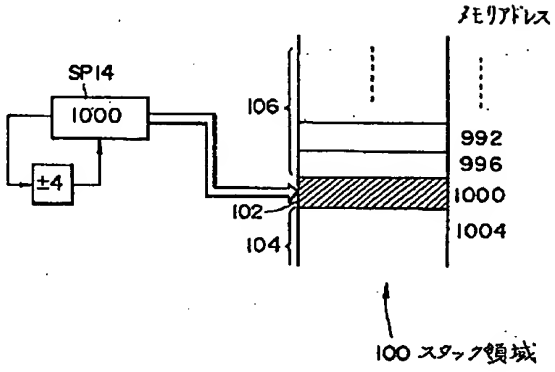
【図1】



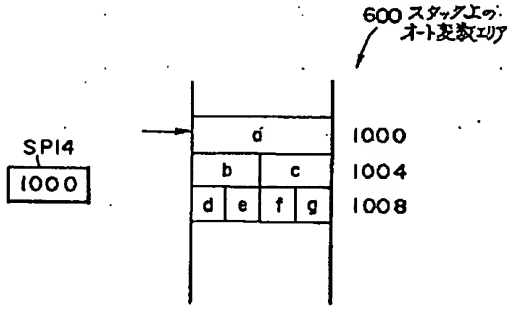
【図2】



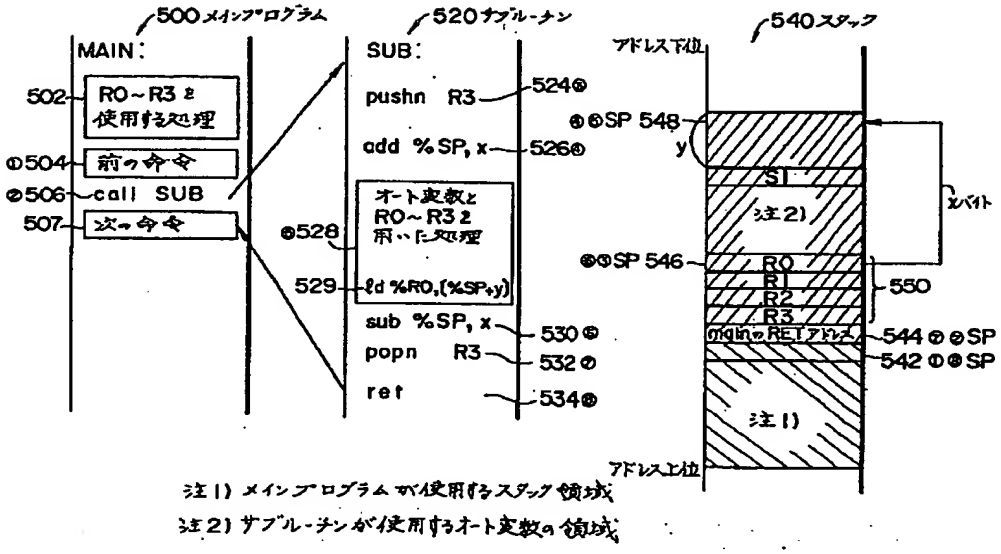
【図3】



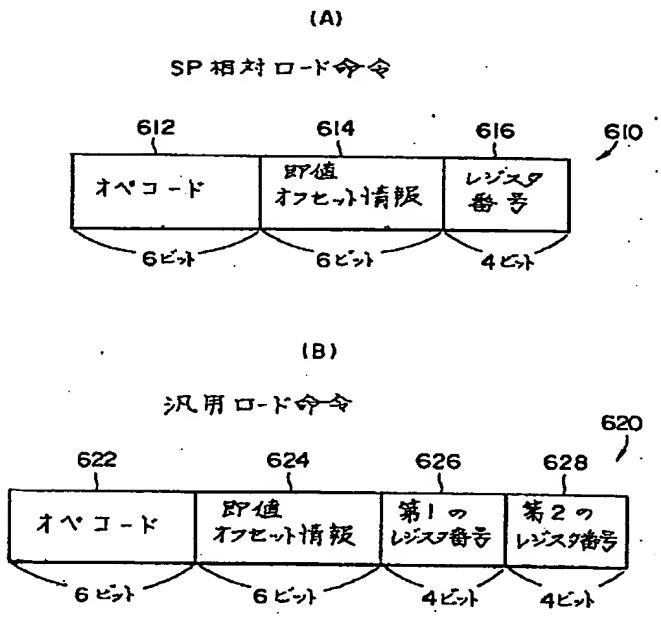
【図5】



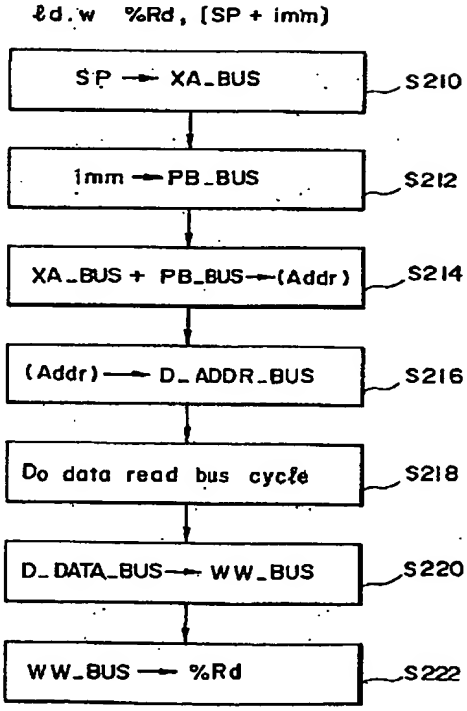
【図4】


500 メインプログラム
MAIN:
502 R0～R3を使用する処理
①504 前の命令
②506 call SUB
507 次の命令
520 サブルーチン
SUB:
pushn R3 524③
add %SP, x 526④
⑤528 オート変数とR0～R3を用いた処理
529 ld %R0,(%SP,y)
sub %SP, x 530⑤
popn R3 532⑦
ret 534⑧
540 スタック
アドレス下位
③④SP 548
y
S1
注2)
xバイト
⑥⑦SP 546
R0
R1
R2
R3
550
mainのRETアドレス
544 ⑦⑧SP
542 ①②SP
注1)
アドレス上位
注1) メインプログラムが使用するスタック領域
注2) サブルーチンが使用するオート変数の領域

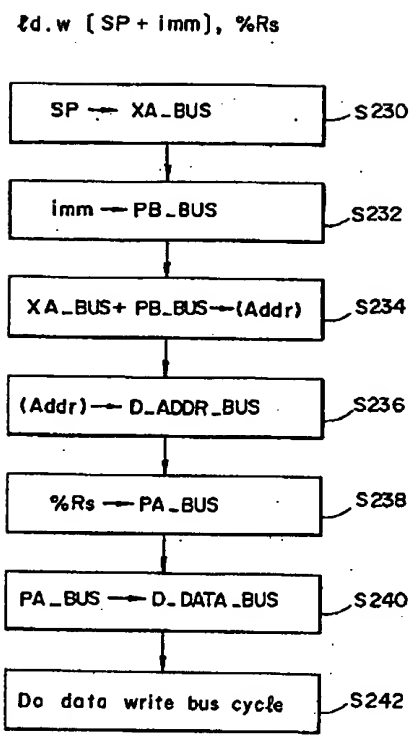
【図6】


(A)
SP相対ロード命令
612
614
616
610
オペコード
即値 オフセット情報
レジスタ番号
6ビット
6ビット
4ビット
(B)
汎用ロード命令
620
622
624
626
628
オペコード
即値 オフセット情報
第1のレジスタ番号
第2のレジスタ番号
6ビット
6ビット
4ビット
4ビット

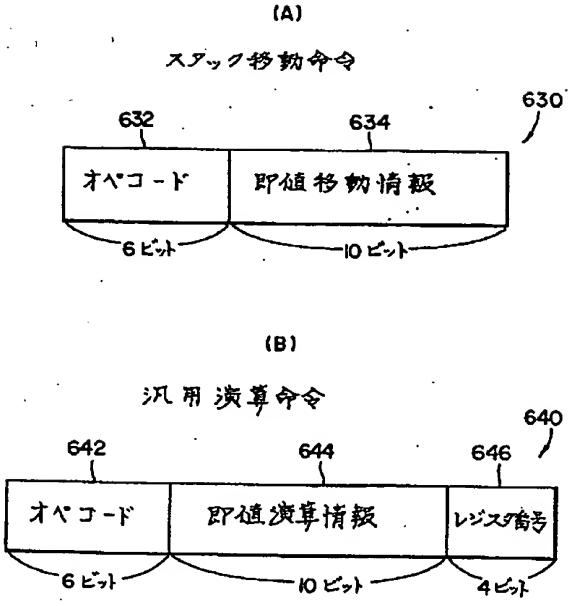
【図7】


ld.w %Rd, [SP + imm]
SP → XA_BUS
S210
imm → PB_BUS
S212
XA_BUS + PB_BUS → (Addr)
S214
(Addr) → D_ADDR_BUS
S216
Do data read bus cycle
S218
D_DATA_BUS → WW_BUS
S220
WW_BUS → %Rd
S222

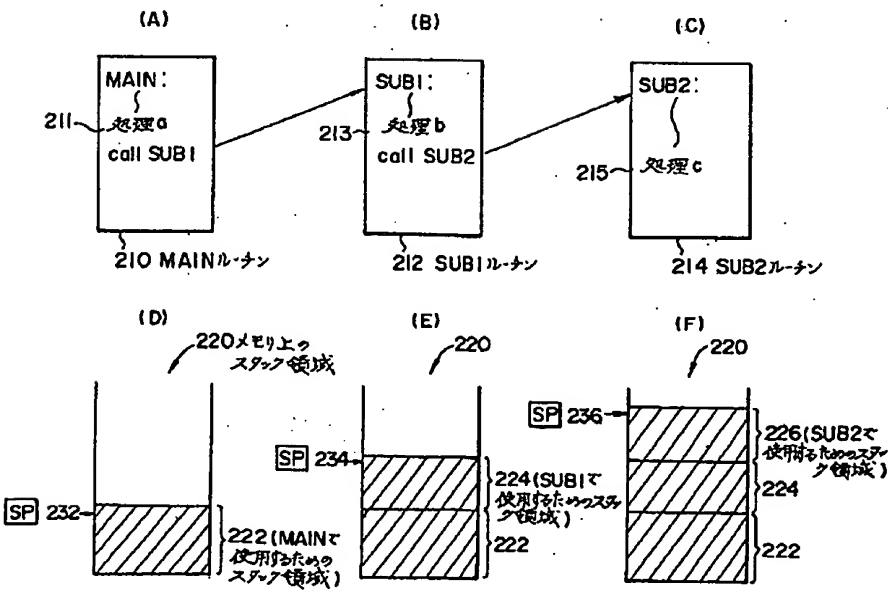
【図8】


ld.w [SP + imm], %Rs
SP → XA_BUS
S230
imm → PB_BUS
S232
XA_BUS+ PB_BUS→(Addr)
S234
(Addr)→ D_ADDR_BUS
S236
%Rs → PA_BUS
S238
PA_BUS → D_DATA_BUS
S240
Do data write bus cycle
S242

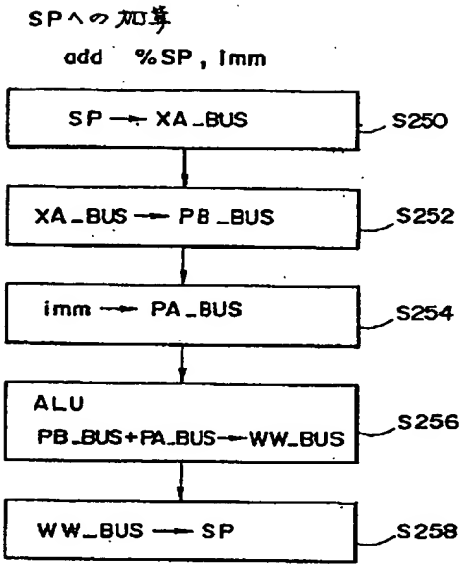
【図10】


(A)
スタック移動命令
630
632
634
オペコード
即値移動情報
6ビット
10ビット
(B)
汎用演算命令
640
642
644
646
オペコード
即値演算情報
レジスタ番号
6ビット
10ビット
4ビット

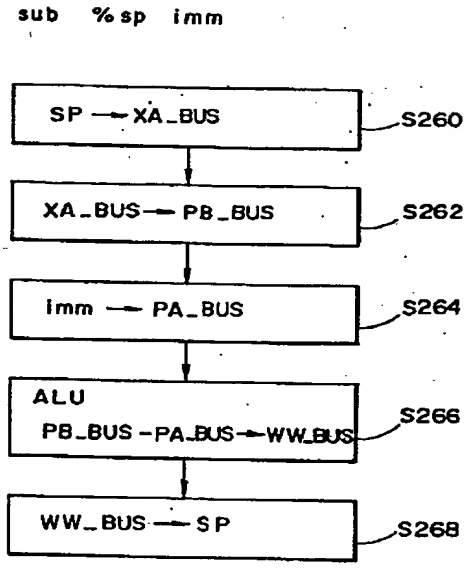
【図9】


(A)
(B)
(C)
MAIN:
211
処理a
call SUB1
SUB1:
213
処理b
call SUB2
SUB2:
215
処理c
210 MAINルーチン
212 SUB1ルーチン
214 SUB2ルーチン
(D)
(E)
(F)
220メモリ上のスタック領域
220
220
SP 232
222(MAINで使用するためのスタック領域)
SP 234
224(SUB1で使用するためのスタック領域)
222
SP 236
226(SUB2で使用するためのスタック領域)
224
222

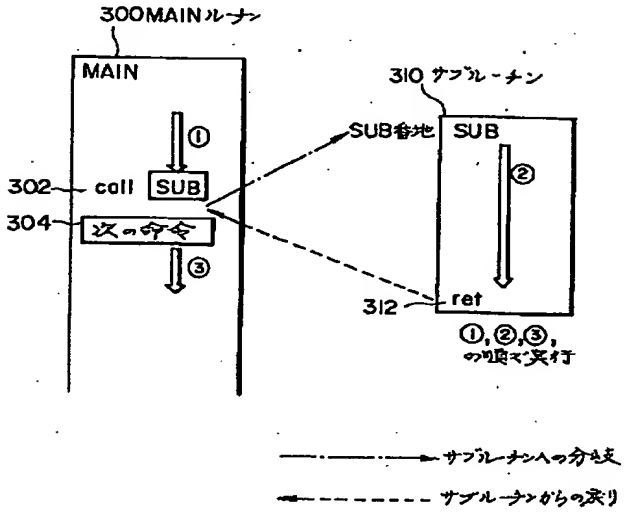
【図11】



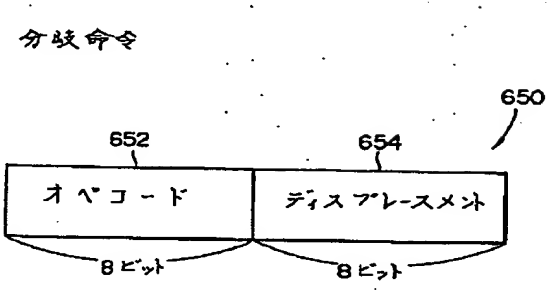
【図12】



【図13】

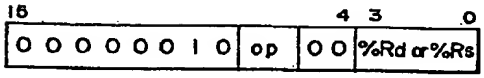


【図14】



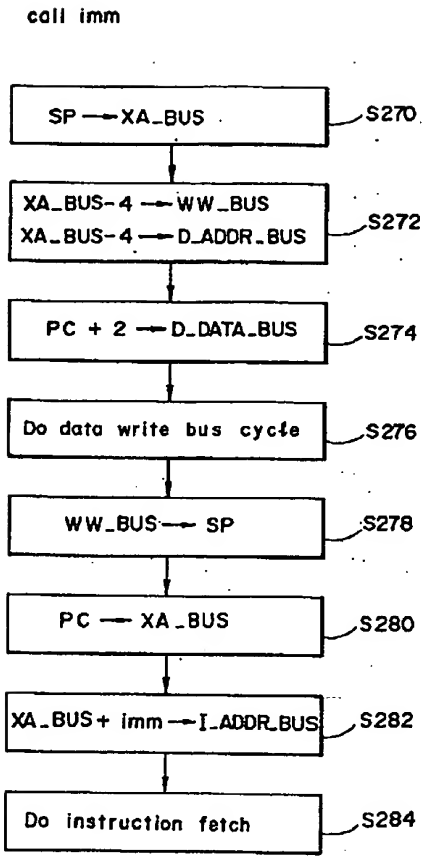
【図19】

pushn 命令, popn 命令のビットマップ

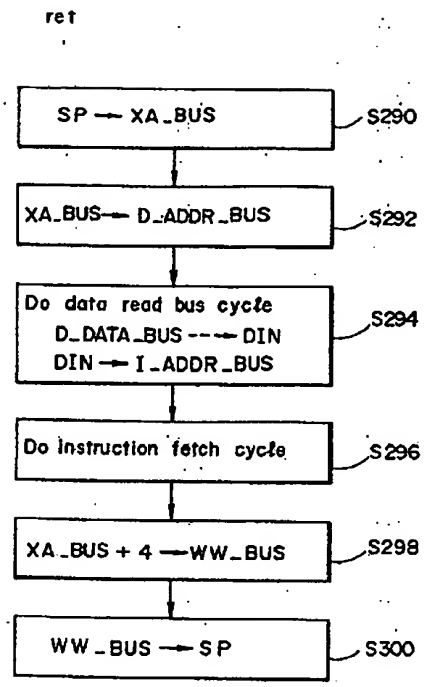


命令	op
pushn	0 0
popn	0 1

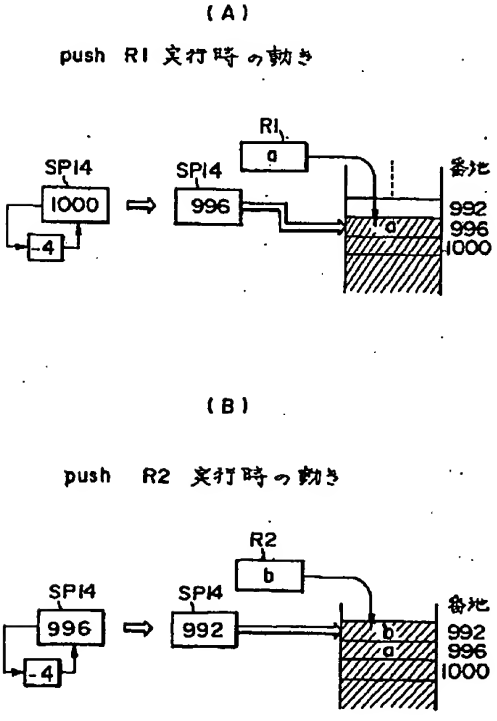
【図15】


call imm
SP → XA_BUS
S270
XA_BUS-4 → WW_BUS
XA_BUS-4 → D_ADDR_BUS
S272
PC + 2 → D_DATA_BUS
S274
Do data write bus cycle
S276
WW_BUS → SP
S278
PC → XA_BUS
S280
XA_BUS+ imm → I_ADDR_BUS
S282
Do instruction fetch
S284

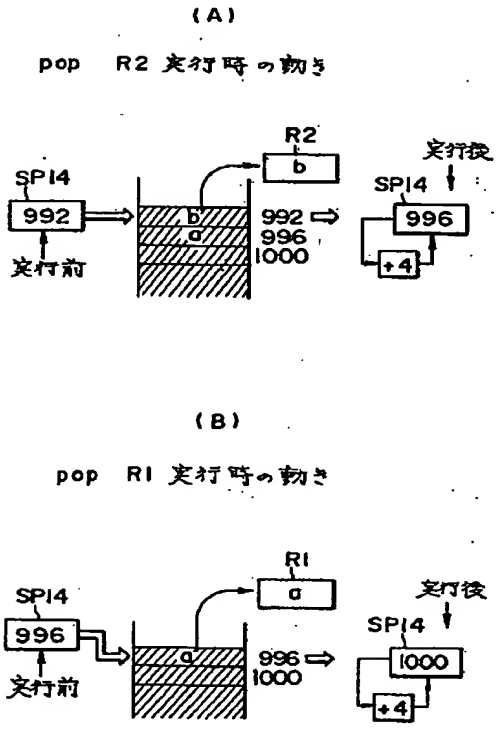
【図16】


ret
SP → XA_BUS
S290
XA_BUS → D_ADDR_BUS
S292
Do data read bus cycle
D_DATA_BUS → DIN
DIN → I_ADDR_BUS
S294
Do instruction fetch cycle
S296
XA_BUS + 4 → WW_BUS
S298
WW_BUS → SP
S300

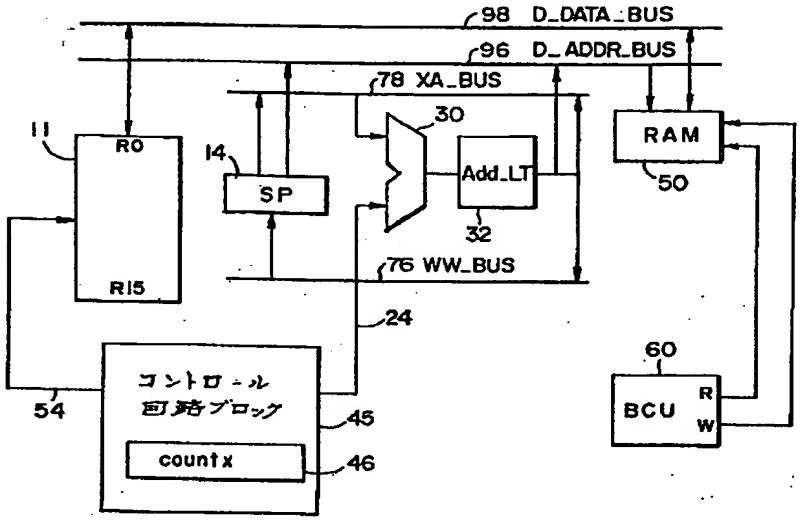
【図17】


(A)
push R1 実行時の動き
SP14
1000
-4
SP14
996
R1
a
番地
992
996
1000
(B)
push R2 実行時の動き
SP14
996
-4
SP14
992
R2
b
番地
992
996
1000

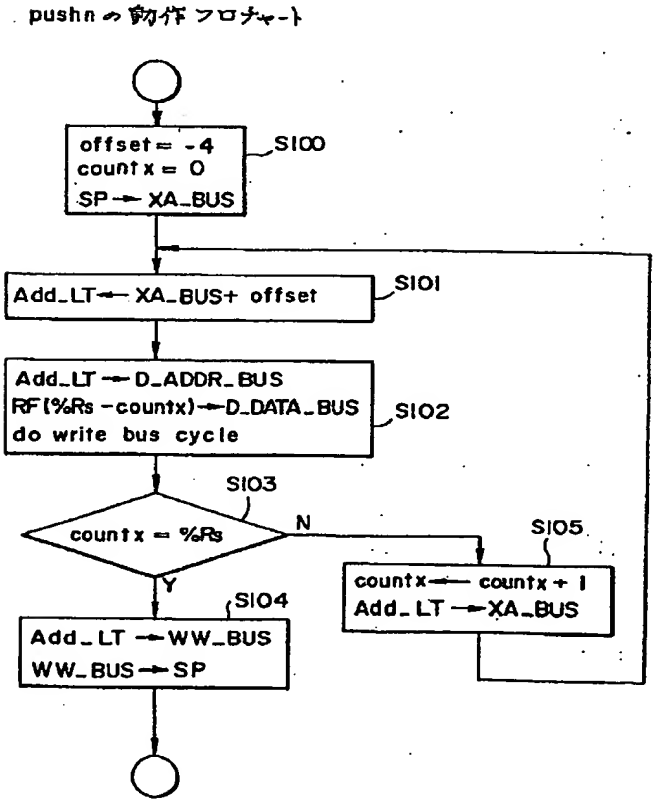
【図18】


(A)
pop R2 実行時の動き
SP14
992
実行前
R2
b
a
992
996
1000
実行後
SP14
996
+4
(B)
pop R1 実行時の動き
SP14
996
実行前
R1
a
996
1000
実行後
SP14
1000
+4

【図20】

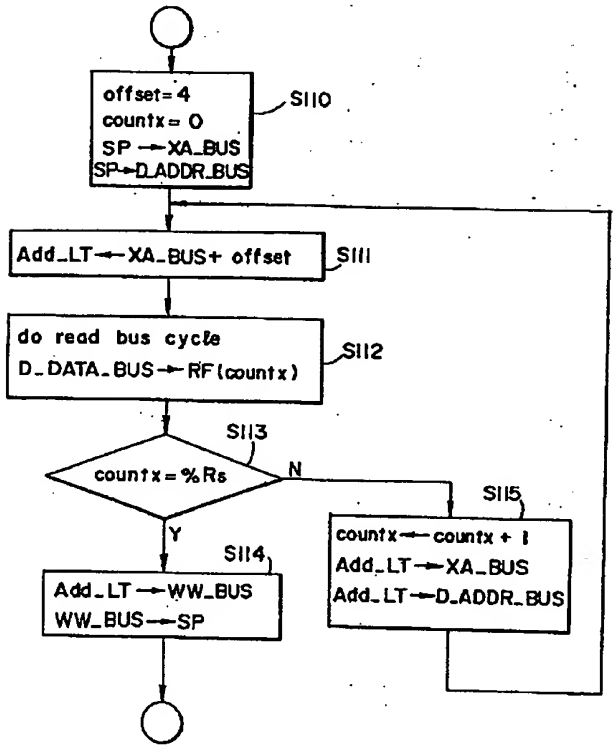

98 D_DATA_BUS
96 D_ADDR_BUS
78 XA_BUS
30
11
R0
14
SP
Add_LT
32
RAM
50
76 WW_BUS
R15
24
60
コントロール
回路ブロック
54
45
countx
46
BCU
R
W

【図21】

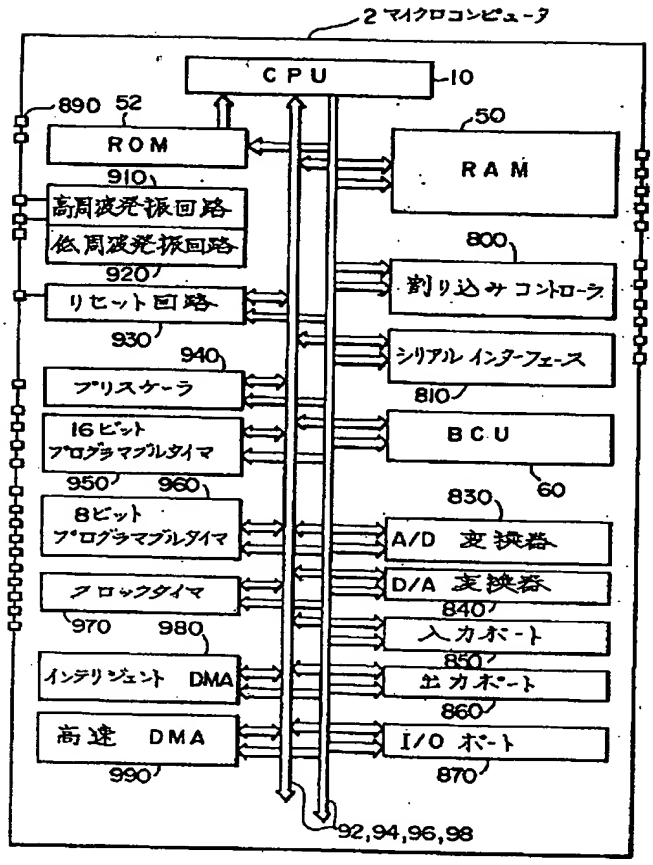

pushn の動作フローチャート
offset = -4
countx = 0
SP → XA_BUS
S100
Add_LT ← XA_BUS+ offset
S101
Add_LT → D_ADDR_BUS
RF(%Rs - countx) → D_DATA_BUS
do write bus cycle
S102
S103
countx = %Rs
N
Y
S105
countx ← countx + 1
Add_LT → XA_BUS
S104
Add_LT → WW_BUS
WW_BUS → SP

【図22】

popnの動作フローチャート



【図２３】



フロントページの続き

(72)発明者　佐藤　比佐夫
長野県諏訪市大和３丁目３番５号　セイコーエプソン株式会社内